VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE ULTRAZVUKOVÝCH MĚNIČŮ A TKÁŇOVÝCH REPREZENTACÍ DO TOOLBOXU K-WAVE

IMPLEMENTATION OF ULTRASOUND TRANSDUCERS AND TISSUE MODELS INTO THE K-WAVE TOOLBOX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN HANZL**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. JIŘÍ JAROŠ, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Práce popisuje návrh rozšíření nástroje k-Wave určeného k modelování šíření ultrazvuku. Cílem rozšíření je minimalizace prosotorové a časové složitosti pomocí alternativního návrhu reprezentace tkání a měničů v simulaci. V práci jsou objasněny základní principy a vlastnosti k-Wave a následně navržena rozšíření a popsána jejich implementace.

Abstract

Extensions to k-Wave toolbox used for ultrasound modelling are described. Aim of extensions is to reduce time and space complexity by presenting alternative representations of tissues and transducers in simulation. This project clarifies basic principles and features of k-Wave, describes design of new representations and finally describes implementation of the suggested extensions.

Klíčová slova

Modelování šíření ultrazvuku, k-Wave, metoda k-space, pseudospektrální metoda, reprezentace tkáně, model ultrazvukového měniče.

Keywords

Ultrasound propagation modelling, k-Wave, k-space method, pseudospectral method, tissue model, ultrasound transducer model.

Citace

HANZL, Martin. *Implementace ultrazvukových měničů a tkáňových reprezentací do toolboxu k-Wave*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Jiří Jaroš, Ph.D.

Implementace ultrazvukových měničů a tkáňových reprezentací do toolboxu k-Wave

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jiřího Jaroše, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Hanzl
23. května 2018

Poděkování

Tato práce vznikla za podpory Ministerstva školství mládeže a tělovýchovy z prostředků účelové podpory Velkých infrastruktur pro výzkum, experimentální vývoj a inovace v rámci projektu „IT4Innovations národní superpočítačové centrum – LM2015070“. Za vedení práce a cenné rady děkuji Doc. Ing. Jiřímu Jarošovi, Ph.D.

Obsah

Kapitola 1

Úvod

Tato práce popisuje rozšíření nástroje k-Wave o nové reprezentace tkání a ultrazvukových měničů. Nástroj k-Wave slouží k modelování šíření ultrazvuku a v praxi nachází uplatnění v medicíně, ale jako volně dostupný software je použitelný i v dalších oblastech. Modelování ultrazvuku je výpočetně velmi náročná úloha, a proto implementace k-Wave cílí na maximální optimalizaci (jak z hlediska výpočetní, tak paměťové náročnosti). Právě za účelem dalšího zlepšení, především v oblasti paměťové náročnosti, mají být nové reprezentace tkání a měničů navrženy.

Začátek textu nabízí shrnutí fungování nástroje k-Wave, jeho použití a osvětlení základů matematického modelu a numerického řešení, které jsou v k-Wave použity. Zvlášť je vyčleněna kapitola rozebírající důvody výpočetní náročnosti řešení. Následující kapitola rozebere stávající návrh programu a jeho vlastnosti a vymezí oblasti pro rozšíření v podobě nových reprezentací měničů a tkání. V závěru kapitoly jsou možná rozšíření navržena a popsána. Kapitola věnující se implementaci postupně popisuje praktickou realizaci všech navržených rozšíření, popisuje detaily stávajícího zdrojového kódu a propojení nově implementovaných rozšíření se stávajícím kódem. Závěrečná kapitola popisuje spuštění simulací nad testovacími daty pro každé rozšíření, hodnotí dosažené výsledky a ukazuje dosažené zlepšení na konkrétních příkladech.

Kapitola 2

Nástroj k-Wave

2.1 Přehled

Nástroj k-Wave, označovaný jinak také jako *toolbox*, je sada programů určená k simulaci šíření ultrazvuku. Umožňuje modelovat šíření ultrazvuku v 1D, 2D i 3D doméně. Podporuje modelování lineárního i nelineárního šíření vln, stejně jako modelování šíření nehomogenním materiálem a absorpci. Díky použitému matematickému modelu a pseudospektrální metodě jeho řešení (viz dále) umožňuje k-Wave simulovat šíření ultrazvuku i při popsaných komplikovaných podmínkách efektivně a s vysokou přesností [2].

Hlavní využití k-Wave nachází při simulaci šíření ultrazvuku v tkáních. Ta se uplatňuje v medicíně při plánování operací pomocí zaostřeného ultrazvuku (FUS – focused ultrasound surgery). Umožňuje přesně určit množství energie absorbované jednotlivými částmi tkáně a nepřímo tak zajistit, aby byly zasaženy právě ty části tkáně, na které má být léčba aplikována.

Přes toto specifické použití je však možné k-Wave používat k modelování šíření ultrazvuku libovolnými médii pro různé praktické úlohy, ve kterých je simulace šíření ultrazvuku potřebná. Za tím účelem k-Wave cílí nejen na efektivitu a přesnost simulace, ale také na maximální jednoduchost a účelnost použitých rozhraní. Součástí toolboxu je řada příkladů a rozsáhlý manuál, který má umožnit uživateli účinně k-Wave použít i bez znalosti detailů simulační metody nebo implementace. Celý projekt je distribuován jako open-source pod licencí LGPL.

Hlavní implementace k-Wave je v prostředí MATLAB, kde je realizována jako knihovna simulačních funkcí společně s definicí tříd, nad kterými simulační funkce pracují. V případě datově nejrozsáhlejších simulací – tedy simulací rozsáhlejších 3D domén – se implementace v MATLABu, coby interpretovaném jazyce, stává faktorem limitujícím výkon. Pro tyto účely je nástroj doplněn C++ implementací 3D simulační funkce, která je vysoce optimalizovaná a využívá prostředí OpenMP pro paralelizaci.

Implementace v C++ představuje samostatný program, který s MATLAB verzí komunikuje skrze datové soubory formátu HDF5. Pro použití C++ verze jsou v MATLABu připraveny doplňující funkce, které zajišťují konverzi a uložení datové reprezentace v MATLABu do HDF5 souboru, spuštění C++ programu nad tímto souborem a převzetí výsledných dat ze simulace, opět přes HDF5 soubor.

Díky použití otevřeného formátu HDF5 pro vstupní a výstupní data se v praxi používá C++ implementace i samostatně. Prostředí MATLAB, coby proprietární software, nemusí být pro koncového uživatele dostupné, případně nemusí být pro daný účel použitelné z právních důvodů. V takovém případě lze použít volně dostupné implementace knihovny

pro práci s HDF5 soubory. Tato knihovna je implementována v jazyce C a jsou k dispozici rozhraní pro její použití v řadě dalších prostředí, včetně jazyků Java, již zmíněný MATLAB, Python, R a mnohé další. Díky tomu lze C++ implementaci k-Wave použít i z těchto prostředí obdobně jako z MATLABu. Nevýhodou je absence obalujících funkcí z MATLAB implementace, dokumentace k-Wave ale nabízí vyčerpávající popis formátu vstupních i výstupních souborů, a použití v jiných prostředích je tak stále relativně snadné.

2.2 Způsob výpočtu

Ačkoli matematický model, který k-Wave realizuje, ani způsob jeho numerického řešení, nejsou stěžejní oblastí této práce, uveďme zde alespoň jejich základní varianty a vlastnosti. Budou díky nim představeny základní pojmy, s kterými další text pracuje, a také vlastnosti výpočtu, které udávají požadované výpočetní operace. Především tyto oblasti jsou pak zohledněny v rámci návrhu a implementace.

Simulace v k-Wave je postavena na základě numerického řešení soustavy parciálních diferenciálních rovnic prvního řádu. Tyto rovnice udávají vztahy pro zachování hybnosti 2.1, zachování hmotnosti 2.2 a vztah mezi tlakem a hustotou 2.3. V základní variantě (tedy v případě homogenního média beze ztrát) jsou tyto vztahy následující [1, 2]:

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho_0}\nabla p \tag{2.1}$$

$$\frac{\partial \rho}{\partial t} = -\rho_0 \nabla \cdot \mathbf{u} \tag{2.2}$$

$$p = c_0^2 \rho \tag{2.3}$$

Kde $\mathbf{u}$ udává rychlost částice, p akustický tlak, ρ akustickou hustotu, ρ_0 klidovou hustotu prostředí a c_0 rychlost zvuku. Výše uvedené rovnice platí pro nejjednodušší případ. Pakliže zavedeme nehomogenní médium a další možnosti, které k-Wave nabízí, jsou rovnice odpovídajícím způsobem rozšířeny. Přibýt mohou (podle požadovaných vlastností) další parametry, především parametr absorpce α závislý na frekvenci, udávaný hodnotami `alpha_coeff` a `alpha_power`, a parametr nelinearity `BonA` (zdroje nelinearity budou zmíněny dále). Detailní rozbor rozšířených rovnic nabízí dokumentace k-Wave.

Nástroj k-Wave využívá pro řešení výše zmíněných rovnic pseudospektrální metodu *k*-space. Pro ozřejmění jejího základního principu nejdříve uvažujme běžné jednodušší metody. Kritická část výpočtu, tedy určení gradientu v bodě, zde probíhá na základě nejbližších sousedů (v nejjednodušším případě jediného přímého souseda). Aby takový výpočet byl přesný, je nutné na jednu vlnovou délku mít relativně velký počet vzorků, a to i v případě použití metod vyššího řádu. Obzvlášť v případě 3D simulací, kde se každý takovýto nárůst požadovaného detailu vzorkování projeví kubicky, to představuje významný problém. V praxi pak doména snadno nabývá rozměrů, jaké nejsme schopni se stávajícím technickým vybavením účinně řešit (detailněji viz dále).

Aby bylo možné i při zachování přesnosti potřebný počet vzorků snížit, využívá *k*-space v prostorové doméně Fourierovu kolokační metodu. Při této metodě je provedena nad daty Fourierova transformace, díky níž je možné výsledný gradient odvodit z jednotlivých složek za použití zpětné Fourierovy transformace. Teoreticky je tak možné snížit potřebný počet vzorků až na zhruba tři na vlnovou délku. Navíc, odpovídá-li rozklad pomocí Fourierovy transformace přesně skutečnému signálu, bude i výsledek vypočítaný touto metodou zcela přesný. Fourierovu kolokační metodu popisuje následující vzorec:

$$\frac{\partial f}{\partial x} \approx \mathbb{F}^{-1}\{ik_x\mathbb{F}\{f\}\} \tag{2.4}$$

Stejně tak je nutné kompenzovat chybu, resp. stanovit dostatečně přesnou metodu v časové doméně, což je obecně smyslem třídy metod označovaných *k*-space, nicméně rozsahem již přesahuje prostor této práce. Detailní rozbor numerické metody pro časovou doménu, stejně jako kompletní diskuzi k Fourierově kolokační metodě, poskytuje dokumentace k-Wave.

2.3 Výpočetní náročnost

Před hodnocením výpočetní náročnosti metody *k*-space je vhodné začít nejdřív motivací pro její použití, respektive srovnáním s jednoduššími metodami.

Jak bylo řečeno, pro dosažení uspokojivé přesnosti u jednodušších metod je potřebných více bodů mřížky na jednu vlnovou délku. Aby byla například aproximace vlny pomocí polynomu dostatečně přesná, uvažujme 10 bodů mřížky na vlnovou délku. Ultrazvuk běžně modelovaný pomocí k-Wave má frekvence v řádu MHz, uvažujme 1 MHz. Při rychlosti šíření zvuku médiem 1500 m/s tak získáváme vlnovou délku 1,5 mm. Pro plánování léčby pomocí ultrazvuku je nutné simulovat krychli o hraně například 200 mm, při 10 bodech na vlnovou délku to je 1333 bodů. Už pro uložení jedné takové trojrozměrné matice v jednoduché přesnosti (4 bajty) by bylo potřeba téměř 9 GB.

Při zavedení nelinearity do modelu se ale i tento odhad stává řádově nedostatečný – v nelineárně se chovajícím médiu se při šíření ultrazvuku začnou objevovat vyšší harmonické frekvence, které (pro zachování přesnosti) požadovanou jemnost mřížky násobně zvýší, což po převedení do 3D zvyšuje nároky na úroveň terabajtů. Jak navíc vyplývá z matematického modelu, pro simulaci je nutné těchto matic udržovat v paměti celou řadu, a jak bylo zmíněno, výpočet přestává být (už jen z hlediska prostorové složitosti) na dostupných prostředcích realizovatelný.

Metoda *k*-space tuto složitost výrazně redukuje právě použitím pseudospektrálního přístupu za použití Fourierovy transformace, jak bylo popsáno výš. Protože pro vypočtení použité Fourierovy transformace jsou nutné všechny body v dané ose, označuje se taková metoda také jako *globální*.

Dominantní výpočetní operací této metody je tedy právě Fourierova transformace, která je algoritmem rychlé Fourierovy transformace účinně řešitelná. Potenciálním problémem je vlastnost globality ve všech osách. Tato vlastnost může představovat omezení při paralelizaci, protože při rozdělení problému na části v rámci paralelního výpočetního modelu se nelze jednoduše vyhnout nutnosti synchronizace napříč všemi částmi. Pro příklad uvažujme nejjednodušší rozdělení na části podle jedné osy domény – právě v ose, ve které je doména rozdělena, pak bude nutné pro výpočet Fourierovy transformace získat data od všech ostatních paralelních částí.

O prostorové náročnosti platí, že přestože díky metodě *k*-space je možné rozsah problému dostatečně snížit, stále je možné hledat co nejlepší způsoby uložení a přístupu k simulačním datům. Možný praktický přínos takto optimalizovaného řešení je v možnosti reálně simulovat větší domény. Ačkoli *modelem* dat pro zpracování v rámci numerického řešení jsou matice reálných čísel (s dimenzí odpovídající doméně), jejich faktické uložení v paměti takové být nemusí.

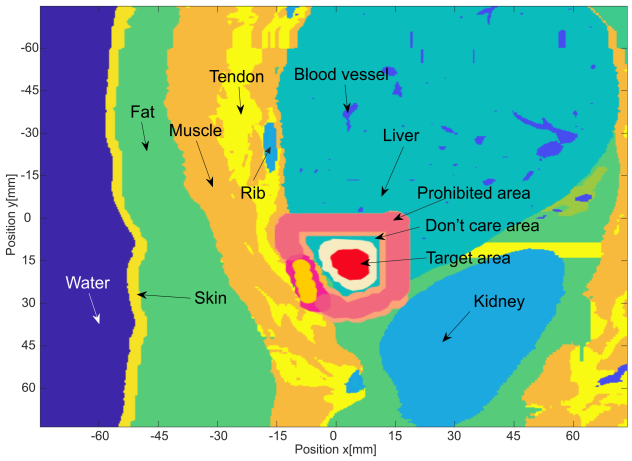
Ať už se podíváme na metody z oblasti zpracování obrazu a signálů, nebo zcela obecné komprese dat, je zřejmé, že možností reprezentace dat je celá řada, a právě jejich volba má

na výslednou prostorovou i časovou náročnost simulace výrazný vliv. Tato volba samozřejmě musí úzce souviset s charakterem a statickými i dynamickými vlastnostmi příslušných matic – některé mohou být po částech homogenní, jiné s vysokou entropií, některé jsou statické po část nebo po celou dobu simulace a podobně. Jako referenční pro hodnocení vlastností různých řešení můžeme uvažovat právě přímé uložení matice coby spojitého pole reálných čísel v požadované přesnosti.

2.4 Reprezentace ultrazvukových měničů a tkání

Předchozí kapitoly popsaly vlastnosti k-Wave obecně. V nich popsané poznatky tak platí pro všechny způsoby použití. Pokud se ale zaměříme na specifické medicínské využití zmiňované v kapitole 2.1, další důležité vlastnosti a požadavky vyplývají z konkrétních praktických situací, které je v daném případě potřeba simulovat.

Prvním důležitým prvkem je popis heterogenního média, v kterém simulace probíhá. V daném případě jde o popis tkáně pacienta. Takový popis se obvykle získává z CT snímku, který je následně ručně anotován. Jednotlivé oblasti na snímku jsou odborníkem vyznačeny a určeny jako různé typy tkání (kost, sval, tkáň určitého orgánu, ...). Pro různé typy tkání jsou známy jejich přibližné fyzikální vlastnosti (rychlost zvuku a další vlastnosti potřebné pro simulaci) a na základě toho je pak vytvořen datový model média pro simulaci.



Obrázek 2.1: Anotovaný popis tkáně

Tento způsob vytváření média není dokonale přesný: Fyzikální vlastnosti tkáně se můžou pro každého jedince mírně lišit, navíc i v různých částech jedné tkáně mohou být mírně odlišné. Stejně tak ruční anotace nebude zcela přesně odpovídat snímku. V praxi se ale ukazuje přesnost jako dostatečná. I proto, že jde o měkké tkáně, a jejich tvar nemusí v době mezi pořízením snímku a aplikací ultrazvukové léčby přesně odpovídat, je chyba vznikající ruční anotací zanedbatelná.

Naopak důležitou, pro pozdější kapitoly této práce užitečnou, vlastností ruční anotace je fakt, že takto vzniklá reprezentace tkáně bude obsahovat jen několik jasně určených druhů tkáně. Rovněž se dá předpokládat, že oblasti jednotlivých tkání budou tvořit spíš větší spojité celky než drobné heterogenní struktury a pravděpodobnost, že by se objevila struktura s detailem na úrovni jednotlivých bodů mřížky je velmi nízká.

Druhým důležitým prvkem jsou ultrazvukové měniče. Ty se v simulaci objevují jak v podobě zdrojů ultrazvuku, tak v podobě senzorů pro záznam fyzikálních veličin v konkrétních bodech mřížky. U zdrojů ultrazvuku platí, že do simulace vstupují obráceným způsobem jako senzory, tedy že pro konkrétní bod mřížky udávají hodnoty akustického tlaku nebo rychlosti v každém časovém kroku simulace.

Obvyklé medicínské zdroje (respektive jejich aktivní část vstupující do simulace) mají běžně podobu podlouhlého pásku rozděleného na více dílčích elementů. Hloubka takového pásu je typicky jeden bod mřížky. Ačkoli je na obrázku měnič rovný, v praxi se setkáme i s měniči zakřivenými. Dalším praktickým příkladem, který budeme uvažovat i u senzorů, je více zdrojů nebo senzorů na části kulové plochy.

Pokud pracujeme s rovným měničem, není zpravidla problém zarovnat simulační doménu tak, aby skutečná poloha zdroje byla kolmá na některou z os a aktivní oblast přesně odpovídala konkrétním bodům mřížky. Vezměme ale v úvahu zakřivený měnič, případně měnič nezarovnaný s simulační mřížkou (který nutně dostaneme při popsaném rozložení více měničů na kulové ploše). V tomto případě je nutné měnič pro simulaci nějak aproximovat.

Stávající jednoduchý přístup je prohlásit za zdroj body mřížky nejblíže sousedící skutečné geometrické reprezentaci zdroje. Takový postup ale nutně vede k nepřesnosti výpočtu, kterou je nutné kompenzovat navýšením detailu mřížky, což nás ovšem vrací k problémům s prostorovou náročností zmiňovaným v kapitole 2.3. Probíhající výzkum [3] ovšem ukazuje, že výrazného zpřesnění je možné dosáhnout i bez navýšení detailu mřížky, pokud aproximujeme zdroj jiným vhodným způsobem.

Zmíněný článek uvádí jako takovou aproximaci následující postup (popis zde je pouze ilustrativní a nebude zacházet do velkého detailu): Použije se geometrická reprezentace zdroje a z ní se vezme přiměřený počet bodů ležících na zdroji. Pro každý bod uvažujeme reprezentaci na mřížce se shodnou granularitou jako simulační mřížka. V daném bodě bude na mřížce jednička a v ostatních bodech nuly. Pokud budeme chtít získat spojitou reprezentaci takového bodového zdroje, na mřížce s danou granularitou to bude funkce sinc s jedničkou v místě daného bodu a procházející nulami v ostatních bodech. Tuto spojitou reprezentaci vezmeme, umístíme zpět na simulační mřížku (s hodnotou jedna ležící přesně na daném bodě zdroje, tedy mimo bod simulační mřížky) a její hodnoty navzorkujeme v bodech simulační mřížky. Výsledkem je reprezentace bodového zdroje na simulační mřížce, kde po sečtení všech bodových reprezentací získáme pole *vah* reprezentující zdroj jako celek.

Takto aproximovaný zdroj ovšem už není možné reprezentovat stávajícím způsobem jako výčet bodů, ve kterých se daný zdroj nachází, ale je potřeba začít pracovat se získanou maticí vah, což bude rozebráno v dalších kapitolách. Zavedení matice vah má navíc potenciální výhodu i pro zohlednění praktických vlastností zdrojů, které obvykle nedokážou produkovat zcela rovnoměrný signál na celé ploše, ale například u okrajů je jejich signál slabší. Obecně vzato, optimální určení vah je stále předmětem výzkumu a jde nad rámec této práce. Samotné zavedení matice vah, ať už budou určovány výš uvedeným způsobem či dále zpřesněny, se ale ukazuje jako potenciálně vhodná úprava s možností zlepšit výsledky simulace.

Jak bylo zmíněno, senzory vstupují do simulace obráceným způsobem jako zdroje. Úvaha o vzorkování bodů ležících mimo mřížku a východisko v podobě matice vah je tedy stejně relevantní pro senzory. I u nich můžeme předpokládat fyzický senzor zabírající více bodů mřížky a produkující jeden výstupní signál, ovšem ležící ne přesně na bodech mřížky a reprezentovaný pomocí matice vah.

U senzorů můžeme sledovat ještě jeden významný praktický příklad související s předchozím, a tím je snímání signálu z velkého počtu míst za účelem přesnější rekonstrukce

jeho šíření. V simulačním prostředí není problém snímat signál z libovolného počtu bodů. Při praktických experimentech ale není vždy možné umístit takové množství senzorů, což může být řešeno například umístěním méně senzorů a opakovaným snímáním, kdy jsou senzory pokaždé mírně posunuty. Tento případ použití je důležitý pro úvahy o reprezentaci a ukládání dat a bude zohledněn v následujících kapitolách.

Kapitola 3

Stávající návrh a možné způsoby rozšíření

3.1 Vstupní data pro simulaci

Z globálního pohledu můžeme na simulaci nahlížet jako na funkci, na jejímž začátku je výchozí stav média, a na konci koncový stav po určitém počtu simulačních kroků. Tomu nakonec odpovídá i rozhraní v MATLABu, tedy obalující funkce, která také udává kompletní výčet potřebných vstupních dat pro simulaci.

Základem je sada *vlastností média*, která udává minimálně skaláry rychlost zvuku a hustotu. V případě komplexnějších simulací pak parametry přibývají: Je možné přidat parametry absorpce nebo nelinearity média, a v případě heterogenního média místo skalárů použít matice. Dalšími nepostradatelnými vstupními daty jsou *parametry simulace* samotné. Mluvíme-li o simulaci jako o funkci provádějící určitý počet kroků a poskytující výsledek v podobě koncového stavu, je samozřejmě tento počet kroků nepominutelným parametrem. Kromě něj jsou to ještě parametry numerického řešení – počet bodů mřížky a jejich vzdálenost v jednotlivých osách a délka časového kroku.

Tím jsou udána data nutná pro simulaci šíření ultrazvukových vln v médiu. Aby ale bylo co simulovat, je potřeba doplnit vlny samotné, tedy přidat *zdroje* zvuku. Pro to nabízí k-Wave čtyři možnosti: První je definice nehomogenního rozložení tlaku na začátku simulace a je definována jednoduše maticí udávající hodnoty tlaku v celém rozsahu domény. Druhou je v čase proměnlivý zdroj tlaku, který se projevuje v rovnici zachování hmotnosti coby zdroj hmotnosti. Je definován maskou bodů, které ke zdroji patří, a sadou hodnot pro každý z bodů, udávající vstupní signál. Třetí možností je zdroj rychlosti částic, vstupující do rovnice zachování hybnosti. Jeho definice je obdobná jako u zdroje tlaku, tedy maska a řídící signál. Rozdílem je pouze to, že řídicí signál má v tomto případě hned tři složky odpovídající jednotlivým osám. Čtvrtou variantou zdroje je reprezentace plochého ultrazvukového měniče, která je de facto zdrojem rychlosti (vstupuje do simulace stejným způsobem) ovšem obsahuje pouze složku rychlosti v ose x (protože jde o plochý zdroj, předpokládá se zde zarovnání s simulační mřížkou, viz kapitola 2.4). Tato definice zdroje obsahuje navíc masku zpoždění, která umožňuje posunout vstupní signál daném bodě o požadovaný počet kroků. Této vlastnosti se využívá pro zaostřování ultrazvukových vln vycházejících z měniče na určitý bod.

Posledním parametrem je definice *senzorů*. I když by simulaci bylo možné na základě již popsaných dat provést, v praxi nám informace koncovém stavu (například v podobě

matice akustického tlaku) nemusí stačit. Naopak můžeme (například při validaci modelu) požadovat data z celého průběhu simulace v určitém bodě (abychom je byli schopni porovnat s praktickým měřením). Protože ukládat kompletní data v průběhu celé simulace by bylo nepraktické (kvůli velikosti dat i kvůli zpomalení plynoucím z nutnosti jejich zápisu), umožňuje k-Wave tyto body explicitně definovat jako senzory. Nabízí k tomu tři možnosti: definici pomocí masky (podobně jako u zdrojů zvuku), definici pomocí n-rozměrných hyperkrychlí definovaných krajními body, nebo definici výčtem souřadnic bodů, v kterých je prováděno měření. Pro implementaci v C++ jsou relevantní uložení ve formě výčtu bodů nebo krychlí definovaných krajními body (zde už není nutno mluvit o hyperkrychli, neboť optimalizovaný kód v C++ je určen k simulaci pouze ve 3D). Měřená vlastnost nebo vlastnosti v tomto případě nejsou součástí definice senzoru ve vstupním souboru, ale parametru při spuštění programu.

Takto definovaná vstupní data jsou uložena a předána simulačnímu programu v podobě HDF5 souboru, který s použitím HDF5 knihovny umožňuje přistupovat k nim transparentně jako k polím reálných čísel (příp. dalších požadovaných typů) bez speciálního zakódování. Potřebné matice nejsou v souboru uloženy hierarchicky, ale pro zjednodušení vždy separátně. Jediné specifikum tohoto uložení udává už samo prostředí MATLAB, kde jsou matice uloženy standardně po sloupcích a indexovány od jedničky. Správné načtení v C++ implementaci, kde jsou matice standardně ukládány po řádcích, zajišťuje implicitně HDF5 knihovna. Jedinou oblastí, ve které se způsob uložení musí zohledni, jsou samotná data v maticích, konkrétně právě matice obsahující indexy použité v definicích zdrojů a senzorů.

3.2 Stávající návrh C++ kódu

Jak bylo řečeno, na C++ implementaci k-Wave lze nahlížet jako na neinteraktivní program, na funkci s jasně danými vstupy a výstupy. Pokud ji rozdělíme na části, je možno identifikovat zhruba následující témata a podúlohy:

V prvé řadě samotnou implementaci simulačního modelu. Ta musí zahrnovat všechny možné varianty simulace s ohledem na požadované parametry a zároveň vyžaduje největší míru optimalizace. Úzce s implementací simulačního modelu souvisí reprezentace a správa matic, nad kterými model pracuje. Ta by měla v maximální možné míře odstínit implementaci simulace od detailů uložení a formátu dat, nicméně při zachování maximální efektivity jak z hlediska paměťové náročnosti na jedné straně, tak z hlediska výkonu simulátoru na straně druhé.

Zmíněné části tvoří hlavní část simulátoru. Připadá k nim ale ještě několik provozních úloh. První z nich je správa HDF5 souborů, jejich zpracování, čtení a zápis. Dalšími jsou zpracování a správa parametrů simulace a případně sledování výkonu během simulace. Pokud počítáme s reprezentací všech dat potřebných k simulaci v paměti a neočekáváme velkou intenzitu zápisů v průběhu simulace, pak žádná z těchto úloh pravděpodobně nebude výkonnostně kritická.

Simulátor je navržen jako samostatná třída. Její primární odpovědností je paralelizované provedení výpočtů rovnic numerického modelu. Tyto výpočty jsou jednotlivě realizovány pomocí samostatných metod, při čemž jednotlivé metody se mohou vyskytovat ve více variantách pro specifické případy simulace (např. lineární a nelineární varianta) a to buď skutečně jako samostatné metody nebo pomocí podmínek v jedné metodě a šablonovacího parametru. Tyto metody jsou označovány jako *kernely*. Kromě nich je odpovědností simulátoru řízení simulace a delegování provozních úloh na ostatní třídy (tj. například načítání dat, alokace paměti a podobně).

Matice potřebné pro simulaci jsou několika základních druhů, podle toho jaká je tvoří data, jaký je význam dat i jaká je jejich role v simulaci. Na tyto typy je možné nahlížet jako na logickou strukturu ve vztahu generalizace–specializace, a tedy ji modelovat pomocí hierarchie tříd. Vlastnosti společné všem maticím shrnuje kořenová abstraktní třída `BaseMatrix`, od které jsou odvozeny základní typy pro matice založené na typu `float` a matice celočíselných indexů potřebné pro zdroje a senzory. Matice typu `float` jsou zobecněním dvou základních typů číselných matic pro simulaci, a to sice matic reálných a komplexních čísel.

Speciálním typem matic jsou komplexní matice zpřístupňující metody pro rychlou Fourierovu transformaci. Tyto třídy v sobě zapouzdřují i části výpočtu simulace a umožňují tak delegovat část operací z hlavní třídy simulátoru. V případě komplexní matice s Fourierovou transformací je to právě rychlá Fourierova transformace a související metody za použití knihovny pro FFT.

Úloha načítání a správy matic je rozdělená mezi třídy matic samotných a speciální třídu kolekce matic. Jednotlivé matice mají na starosti především načtení surových dat pomocí rozhraní pro přístup k HDF5 souboru. Každá ze tříd přistupuje jen k části souboru, která obsahuje data dané matice. Naopak přiřazení částí souboru k maticím, stejně jako koordinaci jejich alokace, dealokace, načítání a zápisu má na starosti třída kolekce matic. Pokud se mluví o načítání a zápisu matic, spadá pod to i načítání a ukládání mezivýsledků, pokud je simulace přerušena v konkrétním časovém bodě a později obnovena. I tady platí rozdělení na načítání na matici, která načítá nebo zapisuje svoje data, a kolekci matic, která čtení a zápis mezivýsledků koordinuje.

Další úlohy už se dají označit jako provozní nebo podpůrné, protože do simulace přímo nevstupují, a zajišťují pouze prostředí pro její chod. Díky tomu, že nejde o kritické části simulace, už není nutné bezpodmínečně dbát na jejich výkon. Je tak možné využívat složitějších abstrakcí a komplexnějších návrhových prostředků a soustředit se na přehlednost a udržovatelnost výsledného návrhu.

První podpůrnou úlohou je už zmiňované čtení a zápis HDF5 souborů. Ačkoli pro formát HDF5 je k dispozici knihovna pro práci s takto uloženými soubory, její rozhraní je zcela obecné, což představuje nevýhodu pro přímé použití ve třídách, které potřebují do těchto souborů zapisovat. Obecnost rozhraní by znemožňovala oddělení záměru od implementace, protože by vyžadovala používání specifických (v daném kontextu však nedůležitých) parametrů a naopak neumožňovala využít přirozených reprezentací objektů v rámci k-Wave. Proto je vhodnější vyčlenit tyto operace do samostatné třídy. Tato třída s výhodou obstarává i další nutné náležitosti při práci se soubory (například správu chyb pomocí výjimek, kterou výchozí HDF5 knihovna neobsahuje) a nabízí polymorfní metody pro snadnou práci s různými typy dat v souborech.

K třídě pro správu souborů se ještě váže abstraktní třída pro zápisové proudy a několik jejích implementací. Účelem těchto tříd je zobecnit zápis různým způsobem vzorkovaných dat, spravovat výstupní vyrovnávací paměti a předzpracovávat data, kde je to nutné. Reálný zápis dat pak zůstává společně se čtením v režii třídy reprezentující HDF5 soubor, jejíž služby zápisový proud využívá.

Zbývajícími částmi jsou správa a zpracování parametrů simulace a zpracování statistik. Správa parametrů je rozdělena mezi dvě třídy, z nichž jedna se stará o zpracování parametrů příkazové řádky a druhá pak spravuje a zajišťuje jednotný přístup k všem parametrům simulace (včetně parametrů z HDF5 souboru). Zpracování statistik není vyčleněno do samostatné třídy, nicméně pro jeho zjednodušení je vytvořena třída pro měření času, která mimo jiné zapouzdřuje odlišný přístup k měření potřebný v paralelním prostředí. Za zmínku stojí

ještě dvě třídy objektů typu *value object*, tedy objektů jejichž identita je určena hodnotou. V dané doméně jsou jimi komplexní číslo a 4D sada rozměrů, použité k určení velikostí matic a případně časového rozměru simulace.

Spolupráce tříd na výpočtu a jeho průběh už z větší části vyplývá z rozdělení odpovědností mezi jednotlivé třídy. Protože jde o jednoúčelový specifický program (de facto podprogram, podíváme-li se na jeho roli v MATLAB implementaci nástroje), není zde mnoho alternativních větví a koordinaci většiny chodu programu může zajistit jediná třída, kterou je hlavní třída simulátoru. Jednotlivými kroky programu jsou postupně načtení parametrů, alokace objektů, načtení dat z HDF5 souborů a samotná simulační smyčka, při čemž v rámci simulační smyčky dojde k zápisu výsledků či mezivýsledků (tím se zajistí, že zápisové operace není nutné duplikovat na více místech).

Zásadnější alternativní tok tvoří jen případné obnovení chodu programu od uloženého mezivýsledku a použití odlišných *kernelů* pro výpočet v případě, že se jedná o specifické typy simulace (tj. vliv nelinearity). Na obnovení výpočtu z mezivýsledku spolupracují třídy simulátoru, kontejneru matic a jednotlivých matic. Alternativní způsoby výpočtu dané různými typy simulace rozlišuje simulátor sám, buď přímo v hlavní smyčce simulace volbou různých *kernelů*, nebo alternativní implementací uvnitř *kernelů* samotných. V těchto případech jde operace klíčové pro výkon simulace, a implementace se tak musí řídit praktickými testy a zohledňovat výkonnostní aspekty v případě nutnosti i na úkor aspektů návrhových.

3.3 Oblasti vhodné pro rozšíření

Podívejme se nejdříve na návrh celkově. Řešení je rozděleno na několik částí, při čemž v předchozí kapitole byly jako kritické části s ohledem na výkon určeny samotné výpočetní *kernely* a potom matice, přístupy k nim a jejich správa. Časová složitost výpočtu je dána především zvolenou numerickou metodou, jejíž dominantní operací jsou Fourierovy transformace (v jednom časovém kroku jich může být nutné provést až 14). Rychlá Fourierova transformace je natolik obecně využívanou operací, že pro její implementaci nabízí řada vysoce optimalizovaných knihoven, ať už proprietárních nebo veřejně dostupných. Nástroj k-Wave jich využívá (je možné ho kompilovat s proprietární knihovnou IMKL nebo otevřenou FFTW) a v tomto ohledu tedy nelze očekávat velký prostor pro optimalizaci.

Zbývajícím faktorem jsou tedy reprezentace matic. Ty jsou ve stávajícím návrhu reprezentovány jako souvislá pole uložených hodnot, většinou s velikostí odpovídající rozměrům celé domény. To (zanedbáme-li vliv vyrovnávací paměti cache) nabízí teoreticky optimální vlastnosti z hlediska rychlosti přístupu (nedochází k vícenásobnému použití nepřímé adresace, k určení indexů není zapotřebí žádných složitějších výpočtů).

Stávající implementaci tedy je možné použít jako referenční z hlediska časové složitosti (přesněji řečeno, z hlediska příspěvku přístupu k datům k celkové časové složitosti). Z hlediska prostorové složitosti je možné ji prohlásit za referenční z opačného úhlu pohledu – jde o výchozí nijak neoptimalizované řešení, pro které (nemají-li data vyloženě náhodné rozdělení) existují efektivnější alternativy. Prostor pro rozšíření je tedy právě v návrhu takových alternativ, porovnání se stávajícím řešením, a ideálně dosažení snížení prostorové složitosti při přijatelném dopadu na celkový výkon. Jak zmiňuje kapitola 2.3, velikost domény, kterou je možné simulovat, má zásadní vliv na praktické použití programu. Redukce prostorové složitosti, díky které by na stejných výpočetních prostředcích bylo možné simulovat větší domény, by byla tedy zvlášť přínosná.

Největší počet použitých matic patří přímo k numerickému řešení. Jde o matice dvou druhů: Jednak samotné matice simulovaných proměnných (tj. rychlost částic, akustická hus-

tota a akustický tlak) a za druhé o matice určitým způsobem parametrizující řešení. Těmito maticemi jsou matice vlastností prostředí, tedy klidová hustota, rychlost zvuku, případně další parametry pro komplikovanější simulace (v případě homogenního prostředí by šlo o skalární hodnoty, uvažujme ale výchozí použití k-Wave pro simulaci léčby ultrazvukem, kde se používá prostředí heterogenní).

Oba typy matic mají některé zajímavé vlastnosti. V případě simulovaných proměnných má rozložení hodnot většinou charakter relativně hladkých vln, čehož využívá především sama simulační metoda (Fourierova transformace nad daty dobře reprezentuje skutečný signál a simulační metoda tak získává přesnost). Zajímavější vlastnosti mají však matice vlastností média. Předně pro všechny tyto matice platí, že jsou po dobu výpočtu konstantní, a jedinými úpravami je případné předzpracování na začátku výpočtu. Druhou zajímavou vlastností jsou zjištění z kapitoly 2.4, že tyto matice vznikají zpravidla ruční anotací, a jsou tak po částech homogenní. Celá doména je tvořena běžně jen jednotkami různých materiálů, a všechny vlastnosti materiálu jsou v oblasti reprezentující určitý materiál shodné. Objevuje se tedy velká redundance, a díky neměnnosti hodnot je zde prostor i pro předzpracování a transformaci dat, pokud by se to ukázalo jako výhodné pro samotný průběh výpočtu. Matice vlastností materiálu jsou proto vhodným kandidátem pro optimalizaci.

Zbytek matic se váže k vstupům a výstupům simulace, respektive k jevům stojícím mimo simulační model a vstupujícím do něj v určitém smyslu z vnějšku. Jde o reprezentace zdrojů zvuku a senzorů. Zdroje zvuku jsou, jak bylo zmíněno dříve, čtyř typů, podle toho, jak vstupují do simulace. Může to být počáteční rozložení tlaku, zdroj tlaku, zdroj rychlosti nebo zdroj rychlosti s maticí zpoždění (pro určité typy měničů). Počáteční rozložení tlaku nové matice do simulace nevnáší, takže můžeme uvažovat jen zbývající tři typy. Tyto zdroje reálně odpovídají dalším typům ultrazvukových měničů.

Ačkoli je v MATLAB rozhraní simulátoru je k dispozici několik způsobů, jak měniče definovat (různě zakřivené měniče, měniče s více oddělenými elementy a podobně), nakonec jsou takové měniče vždy převedeny na základní reprezentaci. Tou je seznam bodů mřížky, které měnič (měniče) tvoří, a řídicí signál pro každý z těchto bodů (jeho hodnota udává hodnoty tlaku nebo rychlosti v daném bodě). Protože řídicí signál je definován po celou dobu simulace, tvoří takto uložené měniče 2D matici o velikosti počet bodů $\times$ délka simulace. V případě zdroje rychlosti tvoří každý bod tři hodnoty signálu pro jednotlivé osy. Měniče tvoří zpravidla velké množství bodů a při delší simulaci je tak potřebné značné množství paměti.

Taková reprezentace měničů z pohledu prostorové složitosti není optimální. Signál v měničích může být periodický nebo může být ve všech bodech měniče shodný (případě shodný s pouhým fázovým posunem). Jak ukazují pomocné MATLAB funkce, definici měničů je často možné zobecnit a reprezentovat například vhodným matematickým modelem vyžadujícím jen několik skalárních parametrů. Všechny tyto vlastnosti ukazují na redundanci dat a dávají možnosti k optimalizaci.

Zvlášť pokud se vrátíme k úvaze o potřebě reprezentace zdrojů neodpovídajících přesně bodům mřížky, byl by tento způsob reprezentace velmi náročný. Díky aproximaci měniče pomocí funkce sinc se počet aktivních bodů násobně zvyšuje a v rámci stávajícího řešení by bylo nutné pro každý z bodů uložit samostatný signál předem vynásobený požadovanou váhou. Ukládání takových dat je ale v podstatě zbytečné, pokud vezmeme v úvahu fakt, že násobení signálu je výkonnostně zanedbatelná operace a bylo by možné ho jednoduše provádět za chodu. Tím by se množství potřebných dat snížilo na jediný signál a jednu matici vah, velikostí odpovídající počtu aktivních bodů zdroje.

V případě senzorů je situace do značné míry podobná jako u měničů. Senzory mohou být definovány výčtem bodů jako měniče, navíc je ale možné je definovat krajními body kvádru. Protože tyto reprezentace nemají časový rozměr, nepředstavuje paměť potřebná k jejich uložení problém. Čistě z hlediska průběhu simulace by nebylo nutné se senzory zabývat; potenciálně neoptimální místo ale představuje část zrcadlící definice řídicích signálů u měničů, tj. ukládání vzorkovaných dat.

Zvlášť při modelování reálných snímačů může být množství bodů na jeden snímač větší, ačkoli reálným výstupem snímače je jen jediný signál. Protože požadovaným výstupem je právě až výsledný signál snímače (který je při stávající implementaci stejně nutné dodatečně odvodit z hodnot všech bodů odpovídajících danému snímači), je ukládání všech hodnot zbytečné. Úpravou dat tak, aby výstupem byl už jen požadovaný výsledný signál, by se dala značně zredukovat velikost výstupního souboru.

I zde platí, že pokud uvažujeme senzor ležící mimo body mřížky a budeme ho aproximovat maticí vah, pak při stávajícím způsobu zápisu by se objem dat ještě násobě navýšil (a výpočet signálu by vyžadoval dodatečné externí zpracování výstupu s nutnou znalostí matice vah) a naopak při zavedení matice vah přímo do simulace se výstup zredukuje na jediný signál. Zpracování vah by probíhalo přímo při zápisu v C++ kódu, bylo tedy velmi efektivní, a nízké paměťové nároky jediného výstupního signálu by umožnily například využít mezipaměti pro jeho ukládání, což by bylo výhodné zejména pro distribuovanou paralelní variantu programu k-Wave.

Poslední oblast pro rozšíření se netýká optimalizace, ale vychází z požadavků praktických případů použití identifikovaných v kapitolách 2.1 a 2.4 a je také nutná jako důsledek případného zavedení matic vah pro ultrazvukové měniče (zdroje i senzory). Stávající implementace pracuje s definicí bodů libovolného počtu zdrojů, resp. senzorů jako s jedinou maticí, kde každému bodu odpovídá jeden vstupní případně výstupní signál. Jedinou výjimku tvoří definice pomocí kvádrů, kde každý kvádr obsahuje větší množství bodů, a tedy různě veliké skupiny signálů jsou do výstupního souboru uloženy jako samostatné matice.

Pokud ale zavedeme více fyzických měničů reprezentovaných jako matice vah, potom může dojít k situaci, kdy jeden bod (vzniklý váhovou aproximací měniče) bude náležet několika různým měničům. V případě zdroje by to znamenalo nutnost složitého předzpracování vah (navíc signál by musel být pro oba zdroje stejný), v případě senzoru by to vylučovalo možnost snímat více signálů úplně. Z toho důvodu je nutné rozšířit vstupní definice i formát zaznamenaných signálů tak, aby bylo možné definovat více měničů nezávisle a nebylo nutné je agregovat, zároveň však při zohlednění výkonnostních požadavků, které se mohou projevit v souvislosti s zápisem výstupu na disk.

3.4 Návrh rozšíření

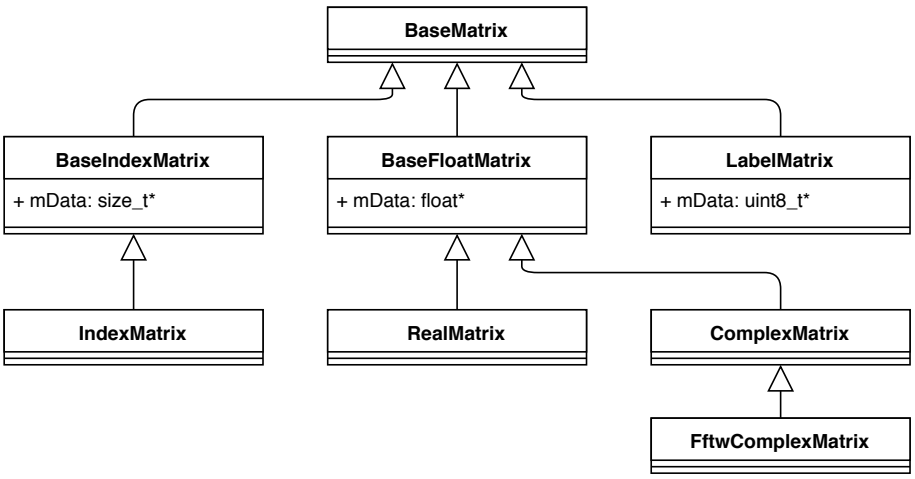
3.4.1 Materiály

První oblastí určenou pro rozšíření je reprezentace materiálů. Jak bylo řečeno, matice reprezentující vlastnosti materiálů vznikají ruční anotací, jsou proto po částech homogenní a zůstávají po dobu chodu simulace neměnné. Každý materiál je definován sadou až sedmi vlastností, každou v přesnosti 32 bitů. Stávající řešení implementuje každou vlastnost jako samostatnou matici, což ale není nutné. Získání vlastností lze rozdělit na dva podproblémy: určení materiálu v daném bodě a následně získání hodnoty konkrétní vlastnosti.

Protože materiálů je malé množství, pro uložení všech hodnot vlastností různých materiálů je potřebná paměť zanedbatelná. Pro identifikaci materiálu v určitém bodě je pak

potřeba oproti stávajícímu řešení výrazně méně místa. Množství materiálů v simulaci pravděpodobně nepřesáhne 10, a na jejich uložení teoreticky stačí 4 bity na bod. Pro prvotní ověření tohoto konceptu lze použít 8 bitů na bod (data tak budou zarovnána a předejde se potřebě bitových operací a adresové aritmetiky). I touto úpravou dojde k výrazné úspoře paměti, a po odladění řešení je možno přejít na 4bitovou reprezentaci.

Taková úprava vyžaduje zavést nové typy matic pro identifikaci materiálů a pro uložení vlastností materiálů. Tyto matice pak stačí použít v dotčených *kernelech* místo původních matic daných vlastností (hustoty atd. . .). Jedinou úpravou získání indexu z matice materiálů a jeho použití pro přístup do matice vlastností místo přímého přístupu do matice dané vlastnosti.



Obrázek 3.1: Hierarchie tříd rozšířená o matici materiálů

Potenciální problém této úpravy představují posunuté matice hustoty, které se využívají pro zpřesnění numerického modelu. Tyto matice vznikají posunem mřížky o polovinu kroku v ose x, y nebo z (jedna matice pro každou osu) a navzorkováním nové hodnoty hustoty média získané lineární aproximací. Jinými slovy, jestliže hodnota v matici hustoty na indexu i odpovídá poloze (x, y, z), pak hodnota v posunuté matici na indexu i odpovídá skutečné poloze $(x + d/2, y, z)$, bavíme-li se o matici posunuté v ose x a vzdálenosti bodů simulační mřížky d.

I pro tyto matice platí, že jsou v průběhu simulace neměnné. Protože ale vznikají lineární interpolací, na rozhraních mezi jednotlivými médii začnou vznikat nové hodnoty hustoty (neodpovídající žádnému z materiálů) a není tak možné tyto interpolované hodnoty uložit stejně jako ostatní vlastnosti materiálu. Vlastnosti spočítané pro krajní body materiálu by v takovém případě neodpovídaly vlastnostem pro vnitřní body materiálu.

Pro přístup k potřebným datům z posunutých matic se nabízí dvě řešení. Prvním je zachování kompatibility s uložením dalších vlastností materiálu zavedením nových vlastností ρ_x, ρ_y a ρ_z. V rámci předzpracování by pak bylo nutné posunuté matice dočasně spočítat a každou kombinaci vlastností uložit jako samostatný pseudomateriál. Tím by počet materiálů řádově narostl (pro každý skutečný materiál by bylo potřeba ukládat více definic materiálů v rámci simulace). Nebylo by pak nejspíš možné uvažovat o navrhované redukci definice materiálu na 4bity, ovšem na 8 bitech by stále byl tento scénář realizovatelný a zachovala by se kompatibilita se stávající definicí *kernelů*.

Druhým přístupem je úplné zrušení posunutých matic a počítání potřebných hodnot za chodu. Toto řešení sice vyžaduje větší úpravy *kernelů*, ovšem dá se předpokládat, že nebude

představovat výkonnostní úzké hrdlo. Výpočet hodnoty lineární aproximací, tedy v daném případě jednoduchým průměrem, můžeme považovat za výkonnostně zanedbatelnou operaci. Pro získání sousedních hodnot hustoty pro průměrování se využije tabulky materiálů, která bude pravděpodobně ve vyrovnávací paměti po celý běh *kernelu*, a matice s indexy materiálu, ke které se již bude přistupovat v osách y a z se střídou, ovšem potřebné body v jednotlivých krocích výpočtu spolu sousedí, a tak ani zde by nemělo hrozit výraznější zpomalení.

Vzhledem k potřebě relativně složitého předzpracování a potenciálně problematického limitu počtu materiálů u první varianty se jeví jako vhodnější řešení počítání hodnot z posunutých matic za chodu. Toto řešení se jeví jako vhodnější i pro budoucí vývoj k-Wave, kdy například při zavedení proměnlivé vzdálenosti bodů mřížky by stejně bylo nutno na výpočet hodnot za chodu přejít. Toto řešení tedy bude zvoleno pro implementaci v další části.

Pokud zhodnotíme navrhované rozšíření materiálů jako celek, nenabídne sice maximální redukci dat (ve které by se dalo pokračovat např. použitím oktalového stromu), ale jeho výhoda spočívá především v zachování přímého a rychlého přístupu k datům (a dopočítávání posunutých hodnot), a tudíž možnosti výrazně zredukovat paměťovou náročnost bez negativního dopadu na výkon. Jakékoli případné další úpravy je pak navíc s výhodou možné provádět nad nově vzniklou maticí materiálů.

3.4.2 Zdroje

Následující oblastí pro rozšíření je první typ reprezentací ultrazvukových měničů, tedy zdroje ultrazvuku. Pro nové reprezentace zdrojů bude potřeba zavést matici vah a z důvodu možného překrytí více zdrojů, když se použije váhová reprezentace, také možnost definovat více nezávislých zdrojů ve vstupním souboru. Z hlediska praktické použitelnosti by každý zdroj měl být nezávislou datovou sadou ve vstupním souboru.

V rámci stávající implementace nemají zdroje ultrazvuku žádnou speciální reprezentaci. Tvoří je indexová matice udávající body zdroje a odpovídající počet vstupních signálů (případně jediný signál, pokud je pro všechny body shodný, a navíc případně matice zpoždění, pokud jde o speciální typ zdroje). Tyto matice jsou uloženy jednoduše stejně jako ostatní vstupní matice v kontejneru matic. Dalšími prvky jsou už jenom definice přepínačů a kód *kernelů* pro zavedení hodnot zdroje do simulace.

V případě více zdrojů s maticí vah takový přístup nelze použít, protože zdrojů může být víc, můžou být různě velké, ale pro každý existuje právě jeden vstupní signál. Je tedy potřeba určit, jak implementaci rozšířit, aby uložení zdrojů v této podobě bylo možné. První možností, v případě, že bychom se snažili o maximální zachování stávajícího uložení, by bylo zachovat ukládání do jedné matice (respektive jedné matice na vlastnost). To by ale znamenalo nutnost rozšířit indexovou matici na 2D strukturu, kde v jednom rozměru by byly indexy bodů a pomocí druhého by se rozlišovaly jednotlivé měniče. Případně by bylo nutné ukládat je za sebe a hranice jednotlivých zdrojů zpracovávat ručně. Rovněž by pak bylo nutné udržovat mapování mezi měniči, vahami a signály a v neposlední řadě by bylo potřeba rozšiřovat kód tak, aby se části matice načítaly z nezávislých datových sad vstupního souboru.

Druhou variantou pro zavedení více zdrojů by bylo uložit matice každého zdroje zvlášť. To by znamenalo rozšíření kontejneru matic tak, aby bylo možné ukládat dynamický počet matic (kontejner doposud zahrnuje vždy jen identifikátor z předem známé množiny jako klíč a právě jednu matici, která může nebo nemusí být přítomna). V tomto případě by

bylo nutné výrazněji upravit kontejner. V podstatě by byla nutná změna všech metod kontejneru, protože nové možnosti uložení matic by muselo zohledňovat načítání, alokace a všechny ostatní operace, které kontejner nad maticemi volá. Verze v *kernelech* by v tomto případě zůstala podobná jako dosud, tedy přímý přístup k potřebným maticím a logika spojující více matic jako jednu entitu zdroje umístěná přímo tam.

Poslední možností by bylo nezasahovat do stávajících definic, ale vytvořit pro zdroje nový kontejner. V takovém případě by se nabízelo i vytvořit třídu přímo reprezentující zdroj, v kontejneru udržovat instance této třídy a správu matic přenést na třídu kontejneru, případně třídy zdrojů. Toto řešení by odpovídalo způsobu, jak jsou nyní uloženy výstupní proudy. Úpravy stávajících definic by byly nutné jen s ohledem na zpětnou kompatibilitu a jejich potenciální převod na nové definice, k čemuž se ještě vrátí kapitola 3.4.4.

Z možných variant bude pro implementaci zvolena poslední zmiňovaná. Přestože nemusí jít o implementačně nejjednodušší variantu, rozdíly mezi variantami nebudou zásadní (každá z variant vyžaduje výraznější úpravy). Naopak výhodou je shrnutí vlastností senzoru pod jeden logický celek a výsledný návrh jasněji reprezentující skutečné objekty domény, oproti roztříštění senzoru do více zcela nezávislých matic. Krom toho bude také nutné upravit způsob načítání (stávající návrh počítá s načítáním datových sad pouze z kořenové skupiny vstupního souboru) a díky implementaci pomocí samostatného kontejneru bude možné logiku týkající se načítání skupin a datových sad držet přímo v něm bez potřeby výrazněji upravovat jiné části kódu.

Při implementaci zdroje coby samostatné třídy je ještě na místě zhodnotit, jestli by u zdrojů nemohla vzniknout podobná typová hierarchie jako u výstupních proudů. Zdroje by se pak (podle struktury a obsažených matic) dělily na zdroje rychlosti, tlaku, váhované zdroje a podobně. Jak ale vyplývá z právě uvedeného výčtu, takové kategorie nejsou vzájemně výlučné (zdroj rychlosti může být váhovaný i neváhovaný, stejně tak u tlaku), navíc pro některé typy zdrojů jsou i tak obsažené matice variabilní (zdroj rychlosti může a nemusí obsahovat matice rychlosti pro všechny tři osy). Z toho důvodu jsou zdroje navrženy jako jedna třída, kde některé matice budou volitelné, a vlastnosti zdroje budou určeny právě přítomností nebo nepřítomností těchto matic.

Závěrečné zavedení nových definic a nové matice vah do výpočtu simulace už vyžaduje minimální úpravy *kernelů*, de facto pouze získání vah (případně nahrazení fixní hodnotou 1, pokud se váhy nepoužívají) a smyčky přes nově vzniklých více zdrojů. Výkonnostní dopad úpravy lze očekávat minimální, navíc v daném *kernelu* se tráví během simulace malé procento času, a neměl by mít tedy na celkový výkon simulace vliv.

Očekávaný přínos tohoto rozšíření je dvojí. Jednak jde o výraznou úsporu paměti, pokud bychom uvažovali příklad zdroje specifikovaného polem vah při původní definici jednoho předpočítaného signálu na bod oproti definici s jediným signálem na celý zdroj a explicitně uloženým polem vah. Druhý přínos je praktický, a sice jasnější definice více zdrojů ve vstupním souboru a díky tomu snadnější tvorba souboru pro uživatele. Přidání obecného pole vah, oproti například geometrické definici zdroje a počítání vah v C++, také umožňuje uživateli přizpůsobovat váhy libovolně podle jeho potřeb a neomezuje tak jeden konkrétní typ jejich výpočtu. To je žádoucí i s ohledem na probíhající výzkum v oblasti, jak zmiňuje kapitola 2.4.

3.4.3 Senzory

Posledním navrhovaným rozšířením je úprava definice ultrazvukových senzorů. Podobně jako u zdrojů ultrazvuku bude cílem zavést matici vah, díky které bude možné přesněji

a efektivně reprezentovat senzory neležící přesně na bodech simulační mřížky, a umožnit v datových souborech (tentokrát na vstupu i na výstupu) definovat jednotlivé zdroje jako samostatné skupiny, respektive datové sady.

Protože logika senzorů v mnohém odpovídá logice zdrojů, v návrhu je možné se některými řešeními z kapitoly 3.4.2 inspirovat. I zde bude potřeba spravovat dynamický počet matic na vstupu a navázat k němu (výstupní) signály a matici vah. Oblasti, které budou výrazněji odlišné, jsou definice výstupů (zápis do výstupního souboru v případě zdrojů nebyl potřeba) a stávající hierarchie tříd výstupních proudů. Ty rámcově odpovídají logickým celkům senzorů, byť zapouzdřují skutečně pouze data týkající se výstupu a vstupní matice jsou na nich nezávislé.

V stávajícím návrhu jsou k dispozici dva typy senzorů: senzor definovaný indexy a senzor definovaný krajními body kvádru. K nim je potřeba přidat nový agregovaný senzor, který bude definován také indexy, ale bude obsahovat navíc matici vah a všechny měřené hodnoty bude na základě ní agregovat do jediného výstupního signálu. Všechny tyto typy budou muset navíc nově umožňovat definice více nezávislých zdrojů, což stávající indexovaná varianta neumožňuje vůbec a stávající definice pomocí kvádrů jen částečně (na výstupu ano, na vstupu však pouze jako jedinou 2D matici, kde jednotlivé senzory jsou rozlišeny odlišnou polohou v ose y). Způsob rozšíření tedy rozebereme následně pro jednotlivé typy zvlášť.

I zde je nutné nejdříve zajistit, jak uložit dynamický počet matic, pokud máme dynamický počet samostatných datových sad reprezentujících více senzorů na vstupu. Na rozdíl od zdrojů se už ale nejeví vhodné zavádět zde samostatnou reprezentaci senzoru a samostatný kontejner. Znamenalo by to totiž nutnost výrazněji upravit i kontejner a třídy pro výstupní proudy (musely by s novou reprezentací senzoru spolupracovat) a zejména samostatná reprezentace senzoru a reprezentace výstupního proudu by se z velké části překrývaly a vytvářely duplicitní reprezentaci senzoru jako logického celku. I přes náročnost zmiňovanou v kapitole 3.4.2 je zde tedy zvoleno rozšíření kontejneru matic takovým způsobem, aby do něj bylo možné více matic pod jeden identifikátor. Zároveň ale musí zachovávat rozhraní pro všechny stávající implementace.

Předpokládejme, že v kontejneru matic máme k dispozici více matic určitého typu (pod určitým identifikátorem) a můžeme je získat jako kolekci. Potřebné vlastnosti senzorů získáme tedy jako několik kolekcí, v kterých každá položka odpovídá vlastnosti jednoho senzoru. Tato data vezmeme jako vstup a můžeme se podívat na výstupní proudy odpovídající jednotlivým typům senzorů a případné rozšíření jejich kontejneru.

Prvním typem je základní senzor definovaný indexy. Umožnit přítomnost více takových senzorů je možné dvojím způsobem. Prvním je inspirovat se už existujícími kvádrovými senzory a mít pro všechny senzory jediný objekt výstupního proudu, který sám rozděluje data do výstupních datových sad. Druhým je vytvořit pro každý senzor samostatný výstupní proud a kolekci výstupních proudů nechat spravovat nadřízený kontejner. Protože definice indexovaného senzoru v původní i nové verzi vstupních souborů je zcela identická (pouze se zavádí vícenásobnost), jeví se jako vhodnější volba ponechat stávající definici výstupního proudu beze změn a umožnit uložit více těchto objektů v kontejneru. Jediné úpravy tedy budou v kontejneru, správné vazbě výstupních proudů na matice s definicemi a zápisů do datových sad mimo kořenovou skupinu výstupního souboru. Ty budou řešeny v rámci implementace.

Druhým typem senzoru je senzor definovaný krajními body kvádru. Při úpravě tohoto senzoru ale nelze postupovat stejně jako v předchozím případě, protože už stávající definice s jistou formou vícenásobnosti počítá. Aby nová definice ve vstupním souboru byla kompatibilní s vícenásobnou definicí ostatních typů a zároveň logika chování senzoru zůstala

zachována, je potřeba jeden nový senzor ve vstupním souboru definovat jako *jediný* kvádr v samostatné datové sadě. Výstup pak zůstává zachován a jeden kvádr na vstupu odpovídá jedné datové sadě na výstupu. Protože stávající reprezentace ale obaluje všechny výstupní datové sady do jediného objektu výstupního proudu, jde de facto mnohem více o změnu definice stávajícího řešení, než o jeho „zmnožení“ zavedením vícenásobné definice senzorů.

Pokud bychom se snažili nové definice realizovat více instancemi výstupního proudu jako u indexovaných senzorů, bylo by nutné kód výstupního proudu výrazně upravit, navíc i s nutným větším doplněním z důvodu potřeby zachování zpětné kompatibility (dále v kapitole 3.4.4). Právě pokud se ale na situaci podíváme jako na pouhou změnu formátu vstupních dat, nabízí se elegantnější řešení: Pokud bychom dokázali načíst nové definice všech krajních bodů kvádrů ze separátních datových sad do jedné matice, reprezentace potřebná pro výstupní proud zůstane nezměněná a úpravy týkající se kontejneru a výstupního proudu budou minimální. Protože velikost definice je fixní (3+3 souřadnice pro dva krajní body kvádru), můžeme na základě počtu senzorů matici požadovaných rozměrů bez problému vytvořit. Data z jednotlivých datových sad na vstupu pak nebude problém načíst do jednotlivých částí matice. Toto řešení bude vyžadovat pouze zavedení alternativní možnosti načítání matice ze vstupního souboru a pro implementaci rozšíření senzorů definovaných krajními body kvádru je vybráno jako vhodnější.

Posledním typem senzoru je nový agregovaný typ. Pro tento typ není potřeba zavádět kompatibilní implementaci se staršími verzemi předchozích senzorů (tj. jediný senzor v kořenové skupině vstupního souboru) a postačí jen implementace odpovídající novým definicím (tj. vícenásobným senzorům). Z toho by se nabízelo použít indexovaný senzor a jen ho vhodným způsobem rozšířit (nejlépe přímo zavedením nové zděděné třídy), ovšem je zde potřeba zohlednit jiné specifikum. Pokud se vrátíme k popisu praktických experimentů z části 2.4, je předpoklad, že k výstupním datům bude potřeba přistupovat jako k celku (přístup se střídou při porovnávání dat reálnými daty senzorů, které se průběžně posouvají). Rovněž pro distribuovanou variantu k-Wave v prostředí MPI by zápis výstupů do samostatných datových sad představoval značnou neefektivitu, jelikož při každém kroku by se do datové sady zapisovala jediná vzorkovaná hodnota. Z tohoto důvodu je žádoucí, aby se u tohoto typu měniče data *všech* měničů agregovala do jediné výstupní datové sady, a to s použitím vyrovnávací paměti, aby k zápisům nedocházelo nepřiměřeně často.

Vzhledem k požadavku na zápis do jedné výstupní datové sady rozšiřování indexy definovaného senzoru není vhodné a nový typ senzoru bude tvořit samostatnou třídu. Tato třída, obdobně jako senzory definované kvádrem, musí obsahovat všechny definice senzorů, aby mohla výsledky zapisovat do jediné datové sady. Samotný princip však bude velmi podobný indexovaným senzorům, pouze při zápisu budou výsledky násobeny vahami a agregovány do jediné výstupní hodnoty.

Úpravy *kernelů* z hlediska senzorů nejsou nutné, protože zápis spravují nezávisle samotné třídy výstupních proudů. V případě agregace dat by tato operace měla být s ohledem na výkon paralelizována – jako vhodnější se zde jeví paralelně zpracovávat jednotlivé senzory, nikoli části matic. Toto řešení je preferované i z hlediska distribuované implementace k-Wave, kde by paralelizace v rámci jednoho senzoru mohla přinášet nutnost nadbytečné komunikace mezi uzly.

Cíle tohoto rozšíření jsou podobné jako v případě zdrojů. Díky váhování bude možné dosáhnout přesnějšího výpočtu signálu senzoru a zavedením výpočtu přímo do simulace se razantně sníží potřebná velikost výstupního souboru. Díky malému objemu dat je pak možné zavést vyrovnávací paměť a snížit počet zápisů na disk. Umožnění definic více senzorů definovaných indexy a více odpovídajících výstupů pak uživateli zjednodušuje práci

s datovými soubory, například v případech, kdy používá více překrývajících se senzorů a v stávající implementaci by je musel spojit do jediné vstupní matice a spojený výstup opět rozdělovat ručně.

3.4.4 Zpětná kompatibilita

Pokud bychom nahlíželi na rozšíření čistě z hlediska funkcí, které k-Wave nabízí, bylo by možné zde s návrhem skončit. Všechna navrhovaná rozšíření buď doplňují funkcionalitu nezávisle na té stávající, nebo tvoří její zobecnění, kdy pomocí nových variant je možné dosáhnout shodného fungování s všemi variantami stávajícími. Protože k-Wave ale zpracovává vstupy a výstupy skrze datové HDF5 soubory, v praxi existuje mnoho experimentů, pro které jsou vstupní soubory už vygenerovány, uloženy, a jejich funkce se musí zachovat. Z tohoto důvodu je potřeba podívat se na návrh všech rozšíření ještě z hlediska zpětné kompatibility.

Stávající implementace k-Wave už na zpětnou kompatibilitu pamatuje, když zavádí v hlavičce každého vstupního souboru číslo verze. Doposud jediným rozšířením zvyšujícím číslo verze bylo zavedení senzorů definovaných krajními body kvádru, o kterých se mluví výš. Toto rozšíření bylo ovšem právě typem rozšíření zavádějící novou funkcionalitu bez zásadních změn té stávající. U zde navrhovaných rozšíření jsou změny zásadnější.

V případě reprezentace materiálů ještě jde o čisté rozšíření, úprava tedy není nutná. Jde o čistě rozšiřující funkci, a tak pro ni stačí speciální hodnota ve vstupním souboru, kde se udá, jestli jde o alternativní reprezentaci materiálů, nebo ne. Výraznější změny stávajícího kódu proběhnou pouze v *kernelech*, kde je možné je efektivně řešit podmínkou a zavedením šablonovacího parametru.

V případě ultrazvukových zdrojů je navrženo zavést samostatnou třídu a kontejner reprezentující zdroj. V tomto případě je potřeba stávající definice zdrojů stále podporovat a bylo by velmi nepraktické duplikovat logiku v třídě zdroje i ve stávající reprezentaci jako samostatné matice spravované kontejnerem matic. Je tedy potřeba přesunout stávající načítání do kontejneru zdrojů. Doplnění třídy zdroje samotného pak už nevyžaduje velké úpravy. Díky tomu, že jednotlivé vlastnosti zdroje (tj. jednotlivé matice) jsou volitelné, stačí správně nastavit jen ty matice, které jsou přítomny v původních definicích a vložit do kontejneru jediný zdroj a zpětná kompatibilita bude zajištěna.

U senzorů definovaných indexy je navrženo použít stávající implementaci, kdy jeden senzor odpovídá jednomu výstupnímu proudu a jedné datové sadě ve výstupním souboru. To situaci značně zjednodušuje, protože jak pro stávající, tak pro novou definici zůstane implementace funkční. Pro původní definice se vloží do kontejneru jediný výstupní proud a pouze se správně naváže na původní matici indexů. Variabilní logika bude potřebná jen při zápisu, kde se výstupní proud bude zapisovat buď do datové sady přímo v kořeni výstupního souboru, nebo do datové sady v podskupině, v případě nové definice.

Senzor definovaný jako kvádr bude mít výstup stejný jako doposud. Tím, že se v návrhu podařilo vyhnout nutnosti úprav tohoto výstupního proudu, je zpětná kompatibilita zajištěna přímo při načítání vstupních matic, a tak je potřeba verzi souboru zohledňovat pouze tam. Definice senzorů s váhovanými výstupy bude v dané verzi nová, a není u ní zpětnou kompatibilitu nutné řešit.

Stávající definice ještě využívají často speciální hodnoty ve vstupním souboru, která určuje, jestli se daná vlastnost používá. V případě nových reprezentací měničů by to znamenalo takovou speciální hodnotu zavádět zvlášť do každého zdroje, resp. senzoru (protože jsou definovány na sobě nezávisle), ovšem tomu musí stejně předcházet detekce přítomnosti

zdroje či senzoru samotného v dané skupině vstupního souboru, a pro nové definice tedy tyto hodnoty už nejsou nutné. Z hlediska kompatibility ale zůstanou zachovány.

Kapitola 4

Implementace

4.1 Rozbor stávající implementace

Zdrojové kódy C++ implementace k-Wave tvoří následující hlavní části: Třídy přímo nebo nepřímo zajišťující simulaci `KSpaceFirstOrder3DSolver`, `MatrixContainer`, `Hdf5File` a třídy implementující různé typy matic odvozené od `BaseMatrix`. Samotnou simulaci implementuje právě třída `KSpaceFirstOrder3DSolver`, v rámci které jsou jako metody implementovány jednotlivé simulační *kernely* – funkce realizující řešení rovnic matematického modelu.

Funkci programu lze na nejvyšší úrovni popsat následujícími kroky (v závorce jsou vždy uvedeny příslušné třídy zodpovědné za danou činnost, případně konkrétní metody):

1. Načítání parametrů (`Parameters`)
2. Alokace paměti (`KSpaceFirstOrder3DSolver`)
    a. Vložení parametrů (jako jsou typy, velikosti, zdroje, ...) do seznamu sdružujícího všechny matice (`MatrixContainer`)
    b. Vytvoření objektů jednotlivých matic (`MatrixContainer`)
3. Načtení dat ze vstupního souboru (`KSpaceFirstOrder3DSolver`)
    a. Načtení počátečních dat pro simulaci (`MatrixContainer`)
    b. Případné načtení dat z posledního uloženého mezivýsledku, pokud jde o pokračování v dříve přerušené simulaci (`MatrixContainer`)
4. Simulace (`KSpaceFirstOrder3DSolver`)
    a. Přípravná fáze
    b. Výpočet jednotlivých kroků simulace (`KSpaceFirstOrder3DSolver`) včetně případného vzorkování dat v každém kroku (`OutputStreamContainer`)
    c. Zápis (mezi)výsledku

Účelem třídy `MatrixContainer` je správa jednotlivých matic nutných pro chod simulace. Tato třída koordinuje jejich alokaci, dealokaci a načítání z HDF5 souborů (a případně i ukládání). Její odpovědností je i vazba konkrétních matic na identifikátory ve vstupním souboru. Samotné načtení dat a jejich správa je už odpovědností tříd implementujících matice, tedy tříd rozšiřujících třídu `BaseMatrix`. Ani ty ale k vstupnímu souboru nepřistupují přímo skrze HDF5 knihovnu. Tento přístup pro ně zajišťuje třída `Hdf5File`, která ze vstupního souboru načítá a validuje surová data určitého typu.

Posledními významnými třídami, které ale už nevstupují přímo do simulace, jsou parametry simulace a třídy vztahující se k výstupním proudům. Třída `Parameters` agreguje všechny potřebné přepínače a konfiguraci simulace. Tato třída implementuje vzor singleton s metodou `getInstance` pro globální přístup. Výstupní proudy odvozené od `BaseOutputStream` jsou objekty propojující definice senzorů (konkrétní matice) s datovými sadami ve výstupním souboru a maticemi vlastností, které se vzorkují. O správu výstupních proudů se stará třída `OutputStreamContainer`, která nabízí podobné operace k `MatrixContainer`u.

Stávající vlastnosti materiálů jsou implementovány pomocí matic s velikostí celé domény, konkrétně `rho0`, `c2`, trojice matic `dtRho0Sgx` resp. `Sgy` a `Sgz` pro posunuté matice hustoty v jednotlivých osách (viz kapitolu 3.4.1) a případně matic `BonA` (pro nelineární šíření) a `absorbTau`, `absorbEta` (pro absorpci). Matice absorpce se nenačítají přímo ze vstupního souboru, ale v přípravné fázi jsou vypočítány na základě vstupní matice `alphaCoeff` a hodnoty `alphaPower`.

V *kernelech* se pro přístup k těmto maticím využívá ukazatelů (přímého přístupu k datům). Důvodem jsou situace, kdy překladač nedokáže optimalizovat přístup k datům pomocí operátoru `[]` (případně jiných nepřímých metod přístupu) a přeloží takový kód do formy volání funkce. Vzhledem k přístupu k každému z prvků matice v každém kroku simulace v takovém případě dochází k úplné degradaci výkonu, a bylo tedy v době implementace nutné zůstávat u řešení pomocí ukazatele.

Zdroje nejsou reprezentovány jako žádná speciální třída. Jsou implementovány pouze jako samostatné matice `pressureSourceInput`, případně `pressureSourceIndex` pro indexy a podobně a nacházejí se v `MatrixContainer`u. U rychlosti jde o matice s prefixem `velocitySource` (v takovém případě jsou tři matice vstupního signálu, každá pro jednu osu) a u reprezentace měniče obsahujícího zpoždění `transducerSource` (u něj je speciálně přítomná matice `delayMask` bez prefixu). O správu matic zdroje se stará pouze `MatrixContainer`, jako u zbytku matic a načítá je podle přepínačů zjištěných ze vstupního souboru v třídě `Parameters`. V případě indexových matic se zohledňuje odlišnost číslování mezi MATLABem a C++ a před začátku výpočtu je potřeba hodnoty v maticích o 1 snížit, což zajišťuje metoda `recomputeIndicesToCPP`, kterou volá v přípravné fázi `KSpaceFirstOrder3DSolver`.

Implementace zdrojů v *kernelech* je stejná jako u materiálů, k datům matice se přistupuje přímo a za použití ukazatele. Veškerá logika týkající se zanesení zdrojů do simulace se nachází přímo v *kernelech*. Odpovídající *kernely* jsou `addPressureSource` u zdroje tlaku, `addTransducerSource` pro zdroj se zpožděním a `addVelocitySource` pro rychlost, v jehož rámci je výpočet ještě realizován voláním funkce `computeVelocitySourceTerm` pro každý rozměr. Paralelní zpracování zdroje je pomocí pragmy `omp parallel for if` omezeno až od zdrojů větších jak 16384 bodů, což má za cíl zohlednit okamžik, kdy zrychlení paralelního výpočtu vyvažuje režii paralelismu.

Za pozornost stojí v *kernelech* specifické použití přepínače `pressureSourceFlag` (případně odpovídajících přepínačů pro další typy zdrojů). Přepínače jsou fakticky implementovány typem `size_t` a je do nich možné uložit i různě vysoké nenulové hodnoty (hodnota se načítá ze vstupního souboru jako skalár a není tak omezená jen na 0 a 1). V stávající implementaci je v hodnotě `pressureSourceFlag` je uložena hodnota, po kolik časových kroků má být případný zdroj aktivní (jinými slovy délka vstupních dat zdroje). V *kernelu* se potom zdroj započítává jenom do dosažení této hodnoty, po jejím dosažení se zdroj zneaktivní a kód *kernelu* se nevykonává.

U senzorů jsou definice implementovány stejně jako u zdrojů, tj. samostatné matice `sensorMaskIndex` nebo `sensorMaskCorners`. Výstup zdrojů zajišťují třídy dědící z abstraktní třídy `BaseOutputStream`, konkrétně `IndexOutputStream` pro zdroje definované výčtem indexů bodů a `CuboidOutputStream` pro kvádry definovaný zdroj. Objekty těchto typů spravuje třída `OutputStreamContainer`. Konkrétně metoda `addStreams` této třídy dostane skrze parametr instanci `MatrixContainer`u, načte si instanci `Parameters` a na základě parametrů simulace sama navytváří požadované výstupní proudy a nastaví jim správné matice definic a snímaných vlastností. Metodu `addStreams` i všechny ostatní týkající se alokace, předzpracování a dalších operací na `OutputStreamContainer`u pak volá třída `KSpaceFirstOrder3DSolver` v jednotlivých fázích simulace.

Konkrétní výstupní proudy pak musí implementovat především tvorbu datových sad a skupin ve vstupním souboru v metodě `create()`, opětovné otevření datových sad v případě obnovy simulace po přerušení `reopen()` a vzorkování `sample()`. Výstupní proud umožňuje definovat *redukční operátor* (například „maximální hodnota“), který pak musí zohledňovat metoda `sample()`. Pokud je použit, data se agregují pomocí tohoto operátoru a zapisují na konci simulace, jinak k zápisu dochází v každém kroku.

4.2 Reprezentace materiálů v MATLABu a jejich uložení

K převedení materiálu na reprezentaci pomocí indexové matice je nutné nejprve zvolit vhodnou reprezentaci v prostředí MATLAB. Tato volba musí korespondovat i se zamýšleným uložením materiálů a indexů ve vstupním HDF5 souboru pro simulaci a nejlépe i s cílovou reprezentací v C++, aby nebylo nutné provádět komplikované předzpracování (pokud je to možné). To je výhodné jak z hlediska výkonu výsledného programu, tak z hlediska snížení komplexity a lepší udržovatelnosti celkově.

Indexová matice (budeme-li pracovat s výchozími 8bitovými indexy) může být reprezentována obdobně jako ostatní matice s velikostí celé domény, pouhou záměnou typu `float` za `unint8`. Typ `uint8` je přímo jedním z vestavěných typů MATLABu. Takto uložené indexy nejsou přímo vázané na MATLAB verzi uložení materiálů (viz dále). Přesto je vhodnější používat indexy od 1, které by odpovídaly přístupu k jednotlivým materiálům v MATLABu. Převést indexaci na hodnoty od nuly je možné při předzpracování v C++. Krom zavedení nového typu do souborů obstarávajících validaci a ukládání objektu `medium` není nutné kód výrazně upravovat.

K reprezentaci samotných vlastností materiálů (tedy hodnot z původních matic hustoty a rychlosti zvuku) existuje více možností. Stávající matice jsou v HDF5 souboru uloženy vždy jednotlivě. To je přirozeným způsobem uložení i z hlediska C++ kódu, aby mohl využívat principu lokality při provádění jednotlivých *kernelů* (porovnejme například se situací, kdy by matice byla jediná, a všechny hodnoty v daném bodě by byly uloženy formou struktury). Stejným způsobem, tedy jako samostatné matice (či de facto vektory), by bylo možné uložit i hodnoty každé z vlastností materiálů (jak v prostředí MATLAB, tak v HDF5 souboru).

Druhou možností je uložit všechny vlastnosti do 2D matice (nebo vektoru struktur). V případě materiálů je, na rozdíl od matic s rozsahem přes celou doménu, tento přístup možný. Hlavní nevýhoda z hlediska C++, tedy možné nabourání principu lokality, tady díky malé celkové velikosti dat nehraje roli. Pokud ovšem vezmeme v úvahu, že všechny potřebné matice materiálů už jsou implementovány v stávající verzi, lze využít přímo tyto matice a pouze zmenšit jejich velikosti na velikost odpovídající počtu materiálů a není potřeba nové struktury zavádět.

K uložení vlastností materiálů jsou tedy použity původní matice. Ty jsou standardním způsobem reprezentovány jak v MATLABu tak v HDF5 souboru, s původní pro simulaci dostatečnou přesností, tedy 32 bitů.

V rámci zavedení nové matice a indexů a změny stávajících matic bylo do MATLAB verze kódu nutné provést následující úpravy: Pro matici indexů bylo nutné připravit definici nového typu mezi stávající definice v souboru `getH5Literals.m`. Ve funkci zajišťujících kontrolu `kspaceFirstOrder_inputChecking.m` byla povolena přítomnost nových matic materiálů a u `kspaceFirstOrder_saveToDisk.m` byla přidána konverze dat v materiálové matici na nový typ. U matic vlastností byly v případě použití materiálové matice upraveny kontroly velikosti a obsahu matic vlastností. Díky zrušení posunutých matic hustoty (viz 3.4.1 a 4.3) bylo také odstraněno generování těchto matic, pokud se používá matice materiálů. V rámci ukládání `writeMatrix.m` je doplněna správná detekce datového typu matice materiálů a přiřazení odpovídajícího typu pro uložení a C++ implementaci.

4.3 Reprezentace materiálů v C++

Stejně jako v MATLABu je nutné v C++ zvolit vhodnou reprezentaci pro indexy a pro materiály. Struktura surových dat je již do značné míry daná. Můžeme ještě zvažovat nějaké předzpracování, ale jak ukázala úvaha v předchozí kapitole, uložení dat v paměti bude nejvhodnější zachovat shodné s uložením v HDF5 souboru. Určité předzpracování stále může být nutné, ale v tomto případě už pouze čistě na základě požadavků algoritmu simulace. Dodatečné předzpracování z hlediska požadavků návrhových a výkonnostních není nutné.

Použitou abstrakci nad daty je možné v C++ volit již volně, nezávisle na paměťové reprezentaci. To samé platí pro způsob přístupu k datům. Zvolená abstrakce by měla respektovat principy dobrého návrhu, tedy mj. vhodně zapouzdřovat data a minimalizovat závislosti mimo dané třídy, ovšem stále s přísným omezením na výkon. Nevhodně zvolená abstrakce by mohla snadno snížit výkon simulace i o desítky procent, což s ohledem na použití programu není přijatelné. Zvolenou optimalizaci ukazuje například stávající implementace materiálů. Je tedy potřeba volit jenom takovou abstrakci, která (ať už díky své podstatě nebo schopnosti kompilátoru optimalizovat) nijak nezvyšuje požadavky na výkon.

Matice indexů je implementována jako třída `LabelMatrix`. Protože se jedná o matici 8bitových čísel, není možné vyjít z žádné ze stávajících obecných matic, a `LabelMatrix` je tak odvozena od základní `BaseMatrix` jako úplně nový typ. Za účelem zachování kompatibility se stávajícím kódem a snadnějšího ověření správnosti výsledků je použit stejný princip přístupu, jako v případě stávajících materiálových matic, tedy použití přes ukazatel a přímý přístup k datům.

Při dalším rozšiřování, tj. redukci indexů na 4 bity, lze této vlastnosti naopak využít, pokud bude bez výkonové penalizace možné využít vektorový operátor `[]`. Stačí pak implementovat tento operátor, aby správně určoval pozici a masku 4 bitového indexu ve fyzických datech, a prostou náhradou ukazatele za referenci na objekt typu `LabelMatrix` bude možné kód otestovat a použít.

V případě vlastností je zachována původní implementace matic, jedinou úpravou je změna rozměrů matic při načítání v třídě `MatrixContainer`. I zde je v rámci *kernelů* zachován přímý přístup k datům přes ukazatel.

Krom nových matic bylo nutné rozšířit i třídy stávající. K načtení matice obsahující nový typ dat z HDF5 souboru je přidána polymorfní varianta metody `ReadCompleteDataset` pro daný typ ve třídě `Hdf5File`. Koordinace načítání nového typu byla doplněna ve třídě `MatrixContainter`, která vyžadovala také doplnění nových definic do seznamů matic a

parametrů `MatrixNames.h` a `Parameters.h`. Krom toho byla doplněna polymorfní varianta metody `readScalarValue<bool>` pro čtení přepínačů v třídě `Hdf5file`. Ostatní přepínače jsou v stávajícím kódu typu `bool` nebo `size_t` podle způsobu načítání. Tato varianta je doplněna, aby se přepínač `labelledMaterialFlag` mohl načíst přímo jako `bool`.

V parametrech přibyl přepínač dostupný přes `getLabelledMaterialFlag`, tento se načte z hodnoty `material_label_flag` ve vstupním souboru. Kromě něj byly doplněny rozměry, které matice s materiály mají při použití daného přepínače. Tyto rozměry jsou dostupné přes `getLabelledDimensionSizes`. V případě použití materiálů se tyto rozměry odvodí z velikosti matice `c0`.

Rozšíření simulátoru zavádí novou metodu pro získání matic `getLabel()` a následnou úpravu *kernelů*, které s materiály pracují. Z výkonnostních důvodů je pro *kernely* byl doplněn šablonovací parametr určující, jestli jde o variantu s materiály. Kde to bylo nutné je pak doplněná podmínka a výběr konkrétní šablonované varianty *kernelu* při volání v nadřízené metodě.

V *kernelech* je vždy načtena dodatečně matice `label` a hustota (příp. jiná vlastnost materiálu) se v případě nastavení šablonovacího parametru získá nepřímým přístupem jako `rho0[label[i]]`. Pro všechny matice vlastností média byl odstraněn atribut `aligned` u direktiv `pragma omp parallel for`. Velikost matic totiž v případě indexované varianty ostatním maticím neodpovídá a přístup do nich je fakticky nahodilý.

K výpočtu hodnot hustoty u posunutých mřížek (staggered grid) přibyla nová inline metoda `computeLabelledDtRho0Sg` s šablonovacím parametrem `dimension`. Podle nastavení parametru metoda vypočte interpolovanou hodnotu hustoty pro mřížku posunutou v ose x, y nebo z. K identifikaci osy přibyl výčet `KSpaceFirstOrder3DSolver::Dimension`. Hodnota se získává lineární interpolací, stejně jako v případě MATLAB implementace. Pokud posunutý bod není definován (dosáhl se okraj mřížky), použije se jen hodnota z existujícího krajního bodu. Výpočet pomocí `computeLabelledDtRho0Sg` se používá pouze při použití materiálů v kernelu `computeVelocityHeterogeneous`.

4.4 Reprezentace a uložení zdrojů

Nové definice zdrojů s váhami jsou implementovány už nezávisle na MATLAB verzi. Jejich definice ve vstupním HDF5 souboru je následující:

```
/p_source/1/mode
/p_source/1/index
/p_source/1/input
/p_source/1/weight
/p_source/1/delay_mask
```

V tomto případě je 1 číslo zdroje (v rámci MATLAB konvence číslováno od jedničky). Vlastnost `mode` udává, jestli se hodnota nahrazuje nebo přičítá a v případě definice zdroje se zpožděním (v kódu označen pouze jako `transducer`) se nepoužívá. Vlastnost `input` je pro zdroje rychlosti reprezentována 3× jako `input_x`, `input_y` a `input_z`. Nově zavedená vlastnost `weight` udává váhy odpovídající bodům v matici `index` a je nepovinná (v případě jejího vynechání tedy výsledek odpovídá původní variantě). Matice `delay_mask` slouží jen pro definice zdroje se zpožděním.

Většina matic je jak je vidět volitelná. Třída `Source` je proto implementována tak, že matice ukládá formou ukazatele a v konstruktoru dostává přes ukazatel názvy matic, které se mají načíst. Pokud se název nezadá (předá se `nullptr`), daná matice se nenačítá.

Pro třídu `Source` bylo alternativní možností zavést typovou hierarchii, ale protože některé matice by musely zůstat i tak volitelné, jsou všechny druhy zdrojů reprezentovány jako jedna třída `Source` a volitelnost matic se používá i k odlišení různých typů zdroje (viz diskuzi v kapitole 3.4.2). Nevýhodou tedy je, že nesprávným použitím se teoreticky mohu pokusit získat referenci na neexistující matici, což ale platí i pro původní `MatrixContainer`.

Třída `SourceContainer` zastřešuje správu všech `Source`, jejich načítání, alokaci a vše související. Tyto operace se pak volají přímo z třídy `kSpace3DSolver` na místech, kde se už obdobně používají `MatrixContainer` a `OutputStreamContainer`. Pro každý typ zdroje je vytvořen samostatný kontejner, aby pak při použití v kernelech bylo možné jednoduše získat iterátor na zdroje daného typu.

V třídě `SourceContainer` je implementována jak původní verze pro soubor do verze 1.1 tak nová verze pro soubor verze 2.0 a vyšší. V případě původních definic zdrojů se kontrolují přepínače načtené ze vstupního souboru, v případě nových definic se `SourceContainer` řídí čistě přítomností skupin a datových sad ve vstupním souboru (mj. proto, že teoreticky mohou různé zdroje stejného typu mít různé vlastnosti).

Kernely se ale pro obě verze řídí přepínači, je tedy nutné i pro novou verzi přepínače řádně nastavit. To z důvodu, že aktuálně funguje vnitřní uložení přepínačů i jako nástroj pro potlačení funkce zdroje od určitého časového kroku, a tato vlastnost byla pro nové definice zachovaná.

K detekci přítomnosti datových sad a možnosti načítat matice i z vnořených skupin byly doplněny metody do `Hdf5File` a přidán nepovinný parametr `h5group` do metody `readData()` v třídě `BaseMatrix` a zděděných třídách.

Protože v třídě `SourceContainer` bylo možné relativně snadno spravovat staré i nové definice zdrojů, je zvolena varianta přesunout i staré definice do třídy `SourceContainer` a jejich původní realizace v třídách `MatrixContainer` a `Parameters` byla odstraněna, aby nebylo nutné je udržovat na dvou místech.

4.5 Uložení senzorů

Stejně jako u zdrojů i reprezentace materiálů je implementována nezávisle na MATLAB verzi a prvním krokem je tedy rozšíření definice ve vstupních HDF5 souborech. V tomto případě se k nim pojí i definice formátu výstupních souborů, které budou doplněny vzápětí. Následující definice vstupů se vztahují pouze k souboru ve verzi 2.0, původní definice jsou k dispozici v dokumentaci k-Wave [2].

```
/sensor/1/type          (1, 1, 1)
/sensor/1/index         (nsens, 1, 1)
/sensor/1/corners       (6, 1, 1)
/sensor/1/weight        (nsens, 1, 1)
```

Číslovka 1 uvádí pořadí senzoru (číslováno od jedničky v rámci MATLAB konvence). Údaj v závorce udává velikost matice, kde hodnota `nsens` je počet bodů senzoru. Položka `type` udává druh senzoru a nahrazuje původní `sensor_mask_type` v kořenové skupině souboru. Odpovídající hodnoty jsou 1 – indexy definovaný senzor, 2 – kvádry definovaný senzor a nově 3 – váhovaný senzor. Matice `index` určuje snímané body mřížky (v případě použití senzorů definovaných indexy a váhovaných senzorů). U váhovaných senzorů se navíc použije matice `weight` s vahami pro odpovídající indexy.

V případě kvádry definovaného senzoru použití vah není povoleno (s ohledem na praktické použití by ani nedávalo smysl) a krom typu se určují jen dva krajní body senzoru ve

formátu $(x_1, y_1, z_1, x_2, y_2, z_2)$ v matici `corners`. Zde je důležitý rozdíl oproti původní definici, kdy nově jedna skupina vstupního souboru (jeden senzor) odpovídá *jednomu* kvádru. Protože ale výstup každého kvádru už teď jde do samostatné datové sady ve výstupním souboru, jeví se toto rozdělení logičtější (a naopak použití více kvádrů jako jeden senzor by způsobovalo nejednoznačné mapování mezi vstupními a výstupními datovými sadami).

Formát výstupních dat je pro každý typ senzoru odlišný a budou uvedeny postupně. Krom rozměrů matice uveden ještě přepínač příkazového řádku, kterým se daného výstupu dosáhne. Pro indexy definované senzory je nově možné definovat senzorů více a výstup je tedy rozdělen do více datových sad následovně:

```
/p/1            (nsens, nt-s+1, 1)      -p
/p_max/1        (nsens, 1, 1)           --p_max
```

U kvádry definovaného senzorů je zachován původní výstup

```
/p/1            (cx, cy, cz, nt-s+1)    -p
/p_max/1        (cx, cy, cz)            --p_max,
```

kde (c_x, c_y, c_z) jsou rozměry kvádru definovaného na vstupu. V případě váhovaných senzorů by definice mohla odpovídat indexy definovanému senzoru (pouze se snížením rozměru matice, protože váhovaný senzor produkuje jen jeden výstupní signál). Na základě diskuze v kapitole 3.4.3 je ale žádoucí, aby všechny váhované senzory produkovaly výstup společně do jediné datové sady. Z toho důvodu je zvolený výstupní formát pro váhované senzory následující:

```
/p              (nelems, nt-s+1, 1)     -p
/p_max          (nelems, 1, 1)          --p_max
```

V tomto případě hodnota `nelems` udává počet senzorů ve vstupním souboru (oproti `nsens`, která v rámci zachování stejné notace s dokumentací k-Wave udává počet bodů senzoru). U všech senzorů je možné používat i další přepínače, například `-u` pro rychlost (výstupy jsou potom ztrojeny pro jednotlivé osy do skupin `ux/1`, `uy/1`, `uz/1`) a další agregační operátory, například `--p_rms` pro kvadratický průměr (úplný výčet přepínačů je stejný jako u původních definic a je k dispozici v dokumentaci k-Wave).

4.6 Reprezentace senzorů v C++

Prvním krokem implementace senzorů do C++ je zpracování vícenásobné definice matic určitého typu ve vstupním souboru. Jak si všímá kapitola 3.4.3, stávající implementace třídy `MatrixContainer` neumožňuje ukládat dynamicky definovaný počet matic. To je dáno použitím vnitřní reprezentace dat jako `std::map`, kde klíč do této mapy je položka z předem definovaného výčtu identifikátorů `MatrixIdx`. Matice pro každou z vlastností může nebo nemusí být v mapě přítomna, podle toho, jak se načte ze vstupního souboru, ovšem není možné uložit matic konkrétního druhu více.

Stávající řešení ukládá do mapy struktury `MatrixRecord`, které obsahují vlastnosti matice (typ, rozměry) a ukazatel na matici samotnou. To proto, aby bylo možné naplnění mapy a alokaci a načítání matic provádět nezávisle. K načtení více matic stejného druhu je doplněna další mapa `mSetContainer`, která neobsahuje jednotlivé `MatrixRecord`y, ale pod každým klíčem vektor `MatrixRecord`ů.

K získání matic uložených ve vektorech slouží nová metoda `getMatrixPtrVector()` (vektory jsou použity, protože do dané kolekce není s ohledem na fungování `MatrixRecord`ů umístit matice přímo). Protože mapa `mSetContainer` obsahuje vektor `MatrixRecord`ů a nikoli matice samotné, pro fungování této metody je doplněna mapa `mPtrVectorContainer`, která obsahuje přímo potřebné vektory ukazatelů a naplní se až po alokaci matic.

Aby se zachovaly funkce třídy `MatrixContainer` a nebylo nutné v metodách duplikovat kód pro matice uložené v `mContainer` a `mSetContainer`, je přidána ještě jedna kolekce: souhrnný vektor `mAllPtrContainer` agregující ukazatele na všechny `MatrixRecord`y. Tento vektor se naplní po prvotní inicializaci dat v obou zmiňovaných mapách. Úprava kódu jednotlivých metod `MatrixContainer`u pak zahrnuje jenom záměnu `mContainer` za `mAllPtrContainer` a úpravy přístupu k objektu kvůli použití ukazatele, ale nebylo třeba nijak měnit jejich logiku.

Dalším krokem je načtení konkrétních matic definujících nové senzory. Protože matice nejsou ze vstupního souboru načítány automaticky všechny (ani by to nebylo vhodné), první úprava je nutná v třídě `Parameters`, která poskytuje při načítání informace o přítomnosti matic, jejich počtu a velikosti. V ní je přidána logika pro načtení požadovaných vlastností ze vstupního souboru (omezena na verzi souboru 2.0 a vyšší) a kromě nastavení potřebných přepínačů také přidána metoda `getSensorSizes()`, která poskytne vektor velikostí senzorů potřebný pro určení velikostí matic při alokaci.

Na základě nových informací v `Parameters` je pak rozšířeno načítání matic senzorů v `MatrixContainer`u. Načítání původních definic je zachováno. Při načítání nových matic je využito nového kontejneru `mSetContainer`. Protože HDF5 knihovna umožňuje přistupovat i k vnořeným datovým sadám přímo, podobně jako u souborového systému čistě zadáním řetězce rozděleného symbolem „/“ (například „/sensor/1/type“), nebyly zde potřeba žádné úpravy načítání a stačilo sestavit správný řetězec udávající cestu přímo při nastavení názvu matice v třídě `MatrixContainer`.

Pro načítání senzorů definovaných kvádrem z návrhu vyplynulo, že by bylo výhodné novou vstupní definici načítat do stávající datové reprezentace, tedy načíst více vstupních datových sad po jednom kvádru do jediné matice agregující všechny kvádry. Pro tento způsob načítání byl zaveden zvláštní přepínač `loadCuboids` do struktury `MatrixRecord` a rozšířena třída `IndexMatrix` o metodu `readCuboids()`, která je schopná požadované načtení z více očíslovaných datových sad provést. V případě nové definice kvádrů se tedy použije jenom jiný způsob načítání, ale `mSetContainer` se nepoužívá a data jsou uložena v původním formátu v `mContainer`.

Tím jsou definice senzorů načteny a zbývá úprava a rozšíření vzorkování a výstupu. Prvním typem senzoru pro rozšíření jsou indexem definované senzory. Pro novou definici těchto senzorů byla v rámci návrhu vybrána varianta více objektů, jeden pro každý senzor. Z toho důvodu byl rozšířen `OutputStreamContainer` obdobně jako `MatrixContainer`, aby dokázal udržovat více senzorů jednoho typu. I zde se používala jako interní reprezentace `std::map`, ovšem protože přístup do mapy pomocí klíče se nikde mimo vkládání prvků nevyužíval, bylo ji možné nahradit za `std::vector`. V případě načítání, alokace a dalších operací to opět znamená minimální úpravy fakticky bez změny logiky. V případě nové definice indexových senzorů je pouze vloženo více výstupních proudů do zmiňovaného vektoru.

V samotné třídě `IndexOutputStream` bylo potřeba zohlednit to, že nyní každá instance třídy `IndexOutputStream` může zapisovat do jiné výstupní datové sady (odlišené čísly). Požadované číslo je přidáno jako nepovinný parametr v konstruktoru třídy. Protože přístup k výstupnímu souboru pracuje už jen s identifikátorem HDF5 zdroje, jediné úpravy se týkají otevření nebo vytvoření datové sady a další úpravy nebyly nutné.

V případě kvádry definovaných zdrojů a třídy `CuboidOutputStream` je situace nejjednodušší, protože formát dat zůstal zachován. Jedinou úpravou je tedy povolení původní logiky i pro případ použití nové definice senzorů. I formát výstupu se v tomto případě zachovává.

Na závěr pro zavedení váhovaných senzorů byla v rámci návrhu zvolena možnost nové třídy `AggregateOutputStream` (oproti podobnosti s `IndexOutputStream` je zde potřeba zápis do jedné výstupní datové sady a možnost sdílení logiky pomocí dědičnosti by byla v podstatě nulová). Tato třída dostává v konstruktoru vektory ukazatelů na indexové a váhové matice a na základě nich pak v metodě `sample()` agreguje data z všech bodů senzoru do jediné hodnoty. Protože množství dat takto vznikající je výrazně nižší než u ostatních senzorů, obsahuje třída navíc vyrovnávací paměť `mBuffer` (a související `mBufferedStepsCount`), do které se data ukládají a k zápisu dochází jen při jejím naplnění. Zbytek kódu se drží stejné logiky jako ostatní implementace třídy `BaseOutputStream`.

Úpravy *kernelů* pro funkci senzorů nejsou potřeba, senzory do nich nijak nevstupují. Jedinými úpravami v třídě `KSpaceFirstOrder3DSolver` je tak potřebný přepočet indexů mezi MATLAB číslováním (zachovaným pro kompatibilitu i u nových definic) a C++ číslováním potřebným pro výpočet.

Kapitola 5

Dosažené výsledky

5.1 Provedení vzorové simulace

K snadnému spuštění a otestování funkčnosti prototypu je v MATLAB verzi do zdrojových souborů doplněn skript `example_labelled.m`, který rozšiřuje simulaci ze stávajícího příkladu `example_ivp_3D_simulation.m` o materiálové matice. Konfiguračními proměnnými lze `labelled`, `nonlinear` a `absorbing` lze upravit chování skriptu, aby generoval původní i novu definici média. Každou z nich je potom možné použít pro novou i upravenou podobou simulátoru a porovnat výsledky.

Po spuštění skriptu v prostředí MATLAB je vytvořen soubor ukázkový soubor s médiem `input_nonlinear_absorbing_labelled.h5` (název bude dle zvolené varianty), který se následně předá jako parametr simulátoru. Simulátor je možné zkompilovat běžně programem `make`, nastavení použitého kompilátoru, volby statického nebo dynamického linkování a volbu použitých knihoven je možno nastavit úpravami uvnitř `Makefile` souboru na místech vyznačených komentáři. Použitý `Makefile` je odladěn pro použití v prostředí superpočítače Salomon, další varianty jsou pak k dispozici v složce `Makefiles`. Detailní popis kompilace spolu s popisem jednotlivých nastavení nabízí dokumentace k-Wave [2]. Simulace se pak spustí následujícím způsobem:

```
./kspaceFirstOrder3D-OMP -i example_input.h5 -o example_output.h5
```

S těmito parametry je možné spustit původní i upravenou verzi simulátoru beze změn. Výsledné soubory by se až na statistické údaje a případné zaokrouhlovací chyby měly shodovat (viz další kapitolu).

5.2 Ověření výkonu a správnosti výsledků tkáňových reprezentací

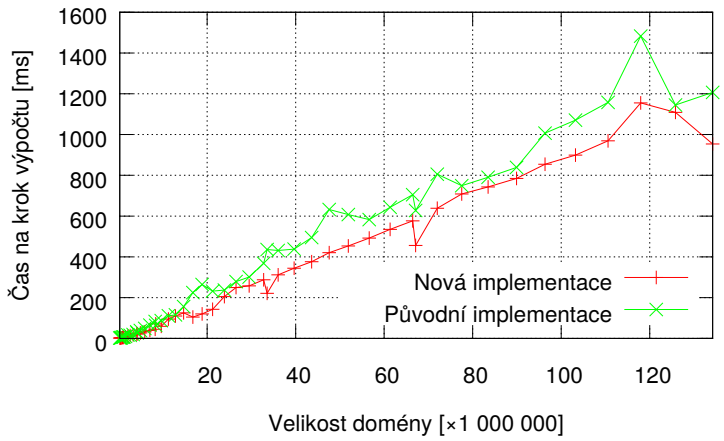
Testování ukazuje, že výsledky upraveného programu se v rámci přijatelné chyby shodují s výsledky původní implementace. Chyba se pohybuje v oblasti do desetiny procenta, v drtivé většině vzorků ale výrazně méně. Jedinou výjimkou jsou hodnoty velmi blízké nule, kde jde ovšem v absolutních hodnotách o tak malé odchylky, že jejich procentuální vyjádření není nutné brát v úvahu.

Teoreticky by se měly hodnoty shodovat přesně, protože nedošlo k změně algoritmu. V rámci testování bylo ověřeno, že použitá hodnota vlastnosti materiálu v *kernelech* získaná pomocí indexové matice je přesně shodná, jako hodnota vlastnosti materiálu získaná

z původní samostatné matice. Rozdíly lze tedy přisoudit činnosti kompilátoru a optimalizacím, v rámci nichž může při změně algoritmu pro získání hodnoty zvolit odlišné pořadí přístupu k jednotlivým hodnotám. Protože u operací s čísly s pohyblivou řádovou čárkou obecně není garantována komutativita, mohou se výsledky mírně lišit. Naměřené odchylky přibližně odpovídají chybě, která by kumulací takové zaokrouhlovací chyby v jednotlivých krocích mohla vzniknout.

Protože implementace nové verze materiálů je zahrnuta i do obalovacích MATLAB funkcí, její otestování je přidáno i do skriptu `kWaveTester` v MATLAB implementaci. Tento skript slouží k automatickému testování shody C++ implementace s MATLAB verzí pro všechny stávající funkce. Je tímto způsobem tedy možné nové reprezentace materiálů otestovat automatizovaně a ověřit, že splňují požadovanou přesnost stejně jako stávající kód.

Výkon nové verze materiálů by se oproti původní implementaci neměl zhoršit. To odpovídá předpokladu, že daná úprava by neměla přinášet požadavky na výpočetní operace ani zpomalení pří přístupu k paměti. Rovněž to koresponduje s faktem, že *kernely* které byly upraveny přispívají k náročnosti celého výpočtu malým dílem. Praktické měření ukazuje dokonce mírné zrychlení oproti referenčním hodnotám, které může souviset s celkovým snížením potřebné paměťové propustnosti, které díky odstranění matic materiálů vzniklo. Naměřené rozdíly ukazuje obrázek 5.1.

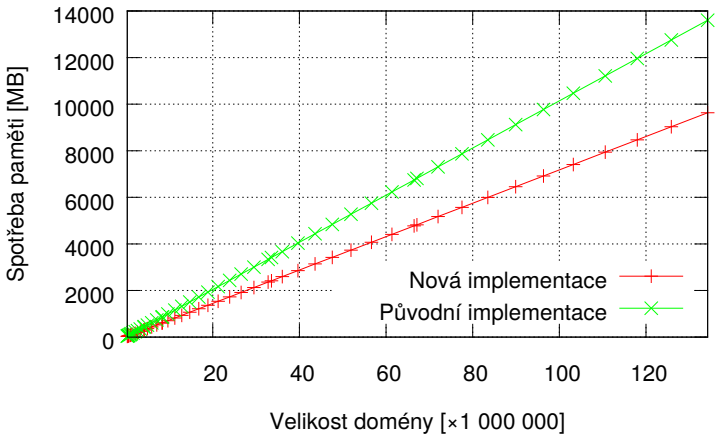


Obrázek 5.1: Časová náročnost v závislosti na velikosti domény

Teoretickou paměťovou náročnost s ohledem na počet matic potřebných k výpočtu rozebírá už dokumentace k-Wave [2]. Pro odhad potřebné paměti uvádí dokumentace následující vzorec:

$$\text{paměť [GB]} \approx \frac{(13+A)NxNyNz + (7+B)\frac{Nx}{2}NyNz}{1024^3/4} + \text{vstup} + \text{výstup} \qquad (5.1)$$

Kde Nx, Ny a Nz udávají velikost domény a parametry A a B jsou určeny dodatečnými vlastnostmi simulace. V rámci implementace došlo k převodu celkem osmi možných matic vlastností na jedinou matici materiálů a vyhledávací tabulku. Pro nejkomplexnější simulaci uvažujme hodnoty $A = 9$ a $B = 0$. Nová matice materiálů (v případě 8bitových indexů) navýší hodnotu A o 0,25 (zabírá celou doménu a její hodnoty zabírají čtvrtinovou paměť oproti zbytku matic). Po dosazení do vzorce dostáváme, že teoretická dosažitelná úspora paměti pomocí tohoto řešení by měla být až 30%.



Obrázek 5.2: Paměťová náročnost v závislosti na velikosti domény

Praktické měření (obrázek 5.2) ukazuje, že očekávané zlepšení se podařilo dosáhnout. Naměřená hodnota úspory je konzistentní napříč velikostmi domény a s rostoucí doménou rychle konverguje k 29,2%. Uvedené měření paměti i výkonu odpovídá kompilaci referenčním způsobem uvedeným v dokumentaci, tj. za použití překladače `icpc` s přepínačem `-O3` na uzlu superpočítače Salomon.

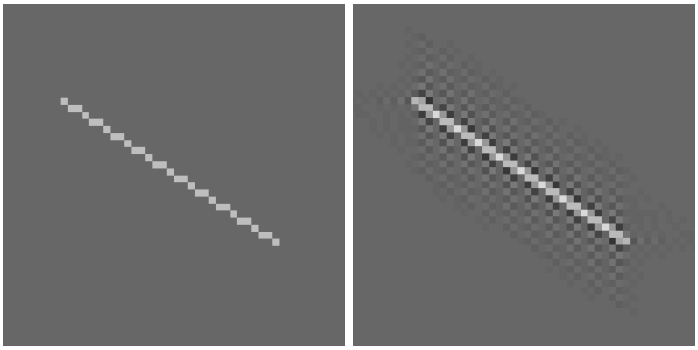
5.3 Otestování a zhodnocení implementace ultrazvukových měničů

V případě měničů (jak zdrojů, tak senzorů) je řešení na MATLAB implementaci nezávislé. Není tedy možné pro něj vytvořit ukázkový MATLAB skript ani přidat do `kWaveTesteru` a je třeba otestovat a vyhodnotit ho jiným způsobem. Pro měniče jsou doplněny skripty v složce `MatlabScripts/InputFileGenerators`, které využívají krom MATLABu i Python a slouží k vygenerování původních a nových (kde je to možné) definic vstupního souboru. Přesný návod na jejich spuštění je v přiloženém souboru `readme.md`.

Daný skript vygeneruje vždy dva soubory (původní a nový). U váhovaných zdrojů vytvoří skript novou reprezentaci tak, že původní zdroj rozdělí na dva, sníží původní hodnotu signálu na polovinu a vloží matici vah s hodnotou 2. Naměřený výstup je tak možné stále porovnat. U váhovaných senzorů se signál agreguje a přímé porovnání tak není možné. Pro senzor je tedy definován jeden element s váhou 2 (vygenerovaný signál by pak měl odpovídat původnímu měření) a zbytek elementů je vložen do druhého senzoru s vahami 1 (výsledek pak odpovídá součtu všech signálů).

Pomocí těchto skriptů byly výsledky změřeny a porovnány nástrojem `h5diff` a v případě kompatibilních definic odpovídá výstup jedna ku jedné, zde dokonce s dokonalou shodou dat (viz srovnání s implementací materiálů, kde díky optimalizaci došlo k drobným zaokrouhlovacím chybám). V případě váhovaných senzorů hodnoty odpovídají očekávaným výsledkům (tj. součtu nebo dvojnásobku) v rámci zaokrouhlovací chyby.

Pro zhodnocení úspory paměti při použití nových definic ultrazvukových měničů se vraťme k příkladu aproximace bodů funkcí sinc zmiňované v kapitole 2.4. Použijeme dvě aproximace měničů, jak je definuje článek [3]:



Obrázek 5.3: Senzor ležící na mřížce a aproximovaný senzor mimo body mřížky

V případě standardní reprezentace měniče přímo v bodech mřížky (na obrázku 5.3 vlevo) je počet aktivních bodů měniče 31. V případě reprezentace měniče vpravo (aproximovaného funkcí sinc) je počet aktivních bodů přibližně 960 (hodnota funkce sinc rychle klesá, a přesně tak záleží na použitém prahování). Pokusme se převést tento odhad do trojrozměrné domény: Pokud bude měnič definován jako čtvercová plocha, v prvním případě to 31^2, tedy 961 bodů. V případě aproximace musíme započíst do hloubky měniče krom samotné hrany měniče ještě body vzniklé aproximací, což je nejméně 8 bodů pro každý směr, celková hloubka aproximované reprezentace bude tedy přibližně 47 bodů. Po vynásobení 960 body v jednom řezu vychází 45120 bodů pro tento způsob reprezentace ve 3D.

Pokud bychom tuto reprezentaci použít v původní implementaci, při běžné simulaci čítající například 1000 kroků by pro uložení signálu pro *jeden* takto definovaný zdroj bylo potřeba 172 MB. Oproti tomu uložení matice vah vyžaduje navíc jen 176 KB a jeden signál pro zdroj 4 KB. Úspora je dle předpokladu přímo úměrná počtu kroků simulace a pokud bychom ji striktně porovnali s původní reprezentací, můžeme mluvit o úspoře přes 99%. Nová váhová reprezentace je proti předgenerovaným signálům zanedbatelně veliká.

V praktickém příkladu umístění měničů na kulovou plochu se navíc předpokládá simulace velkého množství měničů (řádově desítek či stovek). V takovém případě by i z hlediska simulace jako celku mohla data signálu snadno přesáhnout množství dat v simulaci samotné. Díky zavedení váhování se tato data zcela ušetří a bude tak možné ušetřenou paměť využít k provádění větších simulací.

V případě senzorů můžeme u výš uvedeného příkladu mluvit o stejné úspoře při zápisu dat na disk a souvisejícím zrychlením výpočtu, a i zrychlením a zjednodušením jejich zpracování. Protože výstupní soubory neobsahují žádné z velkých matic nutných k simulaci, můžeme se u nich skutečně bavit o zmenšení na zlomek procenta původní velikosti.

Jak u senzorů, tak u měničů lze očekávat při vhodném zvolení vah také významné zpřesnění výsledků simulace. Implementované řešení je dostatečně obecné, aby pomocí něj bylo možné určit váhy i dalšími způsoby na základě dalšího výzkumu, který v oblasti probíhá.

Nezanedbatelnou výhodou nových definic je ale i přehlednější struktura a praktičtější zpracování v odlišných prostředích, než je MATLAB, čehož příkladem jsou nakonec i přiložené pomocné generátory nových definic média v jazyce Python.

5.4 Kombinované možnosti použití

Vezmeme-li implementaci tkáňových reprezentací i váhovaných měničů jako celek, její cíl a využití spočívá především v úspoře paměti bez negativního vlivu na výkon (výkon dosa-

huje dokonce mírného zlepšení). Pro praktické simulace je limitujícím faktorem často právě paměť. Při použití kódu na GPU je množství paměti zřejmým limitem i pro běžné rozměry simulací, jaké vidíme i v příkladech uvedených výš.

V případě výzkumu ultrazvuku s vysokou energií je naopak potřeba simulovat domény ještě výrazně větší. V těchto simulacích roste vliv nelinearity, což vede k výraznému vzniku vyšších harmonických frekvencí a nutnosti zvýšit vzorkování mřížky. Současné příklady se dostávají k hodnotám až 8000 bodů v nejdelší hraně. I v tomto případě hraje absolutní množství spotřebované paměti zásadní roli. Kombinovaná úspora paměti při použití nových reprezentací tkání a měničů v řádu desítek procent tak přispívá k možnosti tyto náročné simulace provádět.

Kapitola 6

Závěr

V rámci diplomové práce byly úspěšně implementovány nové reprezentace ultrazvukových měničů (zdrojů a senzorů) a tkání (materiálů) do programu k-Wave. V rámci ní byly splněny požadované body zadání diplomové práce. U nově implementovaných reprezentací byla ověřena jejich funkčnost a platnost teoretických předpokladů. V rámci zhodnocení byla změřena výkonnost a paměťová úspora po dosažení plné funkčnosti zvoleného řešení. Program byl vyvíjen v prostředí superpočítače Salomon, jak předepisuje zadání.

Text práce poskytuje úvod a představení simulačního modelu včetně numerické metody jeho řešení a rozebírá dále jeho vlastnosti a části, které představují výkonově kritická místa při implementaci. V rámci návrhu požadovaných rozšíření je detailně rozebrán stávající návrh včetně vymezení komunikace s implementací v systému MATLAB, a dále jsou analyzována místa a potenciální možnosti pro rozšíření, včetně vysvětlení a odůvodnění takto zvolených míst. Zvláštní pozornost v odůvodnění volby těchto míst je věnována výpočetní a paměťové náročnosti.

Pro identifikovaná místa jsou následně rozšíření (nové reprezentace tkání a měničů) navržena a popsána. V implementační části je pak popsána stávající implementace a přesně vymezen způsob, jakým jsou k ní doplněny nové vlastnosti implementované v rámci diplomové práce. Závěrečná část shrnuje dosažené vlastnosti výsledného řešení. Krom nich uvádí přínos zvolených řešení pro praktické situace, ve kterých je bude možné použít, a důvody pro zavedení většího rozsahu simulace, kterého díky implementovaným rozšířením bude možné dosáhnout.

Další vývoj je možný v oblasti reprezentací měničů, kde díky stále probíhajícímu výzkumu je zvoleno obecnější řešení. Na základě dalších výsledků je pak možné různě získané váhované reprezentace měničů porovnávat a při dosažení optimálního způsobu určování vah takovýto přístup implementovat přímo do k-Wave. Protože se implementace k-Wave v C++ používá i v jiných prostředích (GPU a distribuovaná paralelní varianta MPI), bezprostřední možností vývoje je převedení implementace i do těchto prostředí.

Literatura

[1] Pierce, A. D.: *Acoustics: An Introduction to its Physical Principles and Applications*. New York: Acoustical Society of America, 1989.

[2] Treeby, B.; Cox, B.; Jaros, J.: *k-Wave User Manual*. Srpen 2016, [Online; navštíveno 12.12.2016].
URL http://www.k-wave.org/manual/k-wave_user_manual_1.1.pdf

[3] Wise, E. S.; Robertson, J. L.; Cox, B. T.; aj.: Staircase-free acoustic sources for grid-based models of wave propagation. In *IEEE International Ultrasonics Symposium*, 2017.
URL http://bug.medphys.ucl.ac.uk/papers/2017-Wise-IEEEIUS.pdf